HOW TO INSTALL FLAGCALC

PETER GLENN

1. Intro

Flagcalc now runs natively on both Linux and Windows (tested on Ubuntu and Pop_OS! 22.04 LTS, alongside very initial and minimal testing in Windows 11). Flagcalc allows you to "query" databases or "workspaces" of several or many millions of relatively small random graphs, for "first order" features such as connectedness, a slew of "measures", criteria and extended notions using a first order language that begins with nestable `FORALL` and `EXISTS` operators, and proceeds to things like `SUM`, `COUNT`, etc. Flagcalc also has some strong features around graph isomorphism classes and automorphisms.

Flagcalc is a command-line tool, invoked such as:

```
./flagcalc -r 10 8 3 -v graphs
```

which chooses three random graphs on ten vertices and outputs their adjacency matrices and edge incidences. As appropriate for a command-line tool, your development environment may be as simple as a Bash terminal window alongside the default Linux text editor (which include syntax highlighting, especially notably parenthesis matching for staying sane around deeply nested queries).

2. Using the Pre-Compiled Binaries

The Github link to releases is

`https://github.com/corralledcode/flagcalc/releases`.

There, the most recent release of binaries (pre-compiled code) is at the bottom of the page. Unzip the appropriate file into a folder (perhaps called "flagcalc"). By "appropriate file" is meant: choose between Windows and Linux, and choose between regular or "CUDA" (the latter only if you have a CUDA GPU, such as provided by NVIDIA). You will see three folders. Note with the latest release, that the scripts all point to the folder called `cmake-build-debug` instead of to the folder `bin`. This is simply an oversight in the release; manually change the first line of each of the six scripts (`.sh`) files in `scripts` to point to `bin` instead.

Date: December 13, 2025.

3. An IDE

Or you can literally compile yourself, and/or open the script files in the standard IDE: JetBrains CLion, available from JetBrains for free for non-commercial use here: `https://www.jetbrains.com/clion/promo`.

(1) One creates a folder `CLionProjects` and downloads the project source code zip file from the Github website (at

`https://github.com/corralledcode/flagcalc/`

where you click the green "Code" button and choose to download zip), then unzips it in the `CLionProjects` folder, next renaming the unzipped folder `flagcalc-main` to `flagcalc`, or
(2) (much preferred), one uses the "New project from a repository" option, with the git URL `https://github.com/corralledcode/flagcalc.git`. This creates the folder `<home>/CLionProjects/flagcalc/` and CLion asks for certain permissions for this folder, but the bulk of the setup is automatic.

There are two general ways to compile the code, either "CUDA-enabled" or "non-CUDA-enabled", depending on whether your host machine has a CUDA GPU (typically from NVIDIA). If you are downloading a pre-compiled binary, please choose from these two options via the particular filename of the file you choose to download (there are four possible files, a grid of Linux vs. Windows, and CUDA vs. non-CUDA). If you want to compile flagcalc yourself, please open the project file `CMakeLists.txt` and choose to set the variable `INTERNALFLAGCALC_CUDA` either `ON` or `OFF`, then click `Tools` menu, `CMake`, then `Reset cache and reload project`, before clicking the "Build" or the "Run" button.

Whether using the CUDA-enabled option or not, if you are compiling yourself then the nvcc.exe compiler must be used (and one can install it even if the machine doesn't have a GPU). Therefore, after installing the CUDA developer's toolit, it is important to edit "settings" under the "File" menu of CLion: please refer to the URL

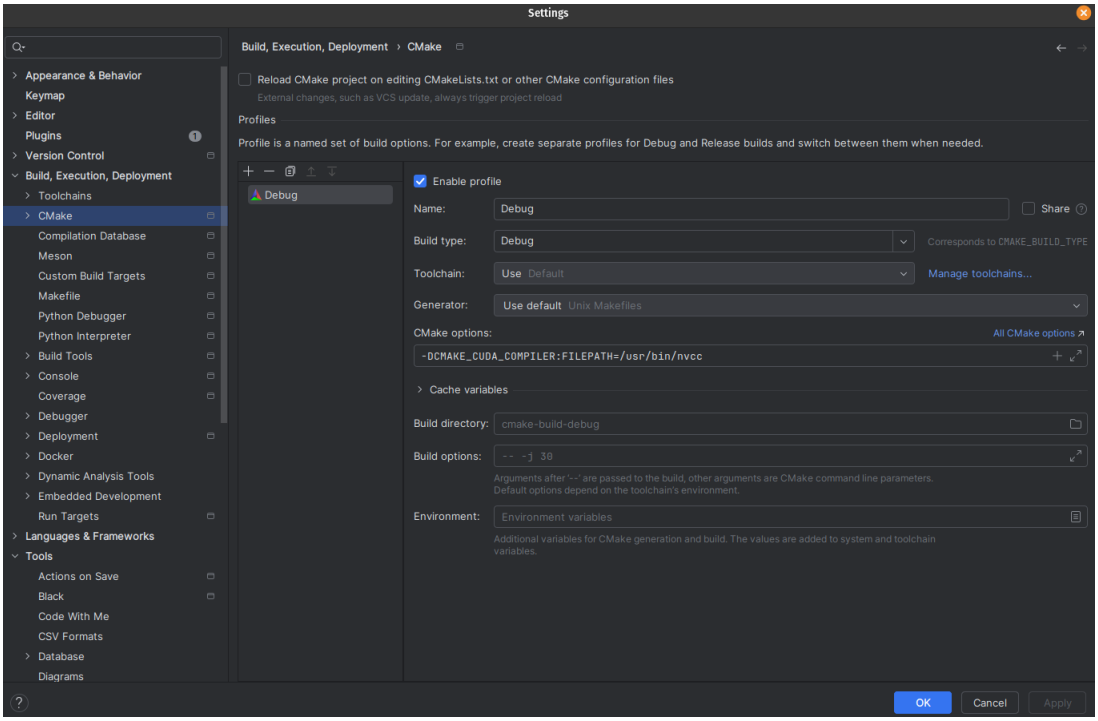
`https://developer.nvidia.com/cuda-downloads`

(where a series of command-line `wgets` and `sudos` are given) for the requisite developer tools from NVIDIA (including nvcc); next, please see

`https://www.jetbrains.com/help/clion/cuda-projects.html`

for instructions about adding all-caps `CMAKE` options to the "CMake options" pointing the CMake tool to the location of `nvcc`.

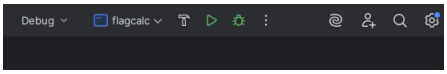
The goal of those two steps is to point to "nvcc" as the compiler as a wrapper for "gcc" or "clang":



Now, if using MS Windows, please jump ahead to the later section, “Specific to Microsoft Windows” for guidelines to this new platform.

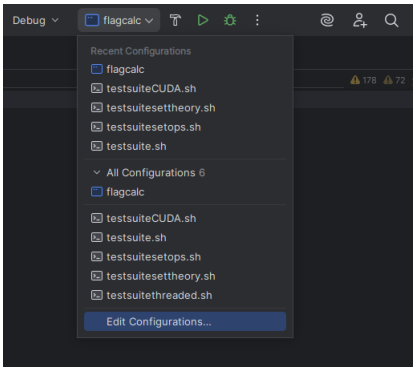
4. A first build

One opens up the project by selecting the “flagcalc” folder just created, and clicks the “build” button (the hammer) in the top right of CLion.



5. Edit run configurations

One can then click the drop-down to the left of the hammer, choose “Edit Configurations”,



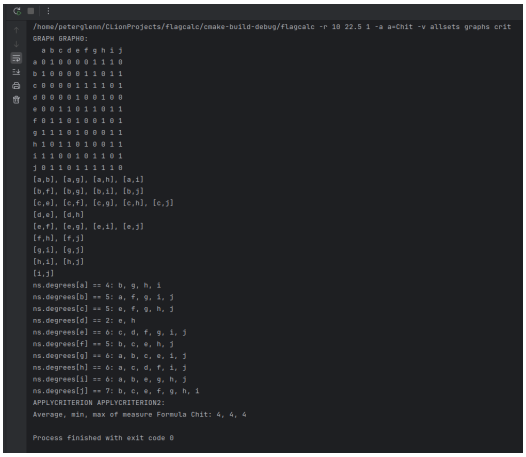
and alter "Program arguments" to consist of what you'd like to pass to the executable by way of a command line argument. Note the "working directory" argument as well needs to be updated to match with where the given package has placed test graph files, if you are using the `-d <filename>` option or are using the `-v i=minimal.cfg` verbosity option (of course you can also choose to move files elsewhere).

For example, to compute a graph's chromatic number one first chooses where to get the graph from, either a file (`-d <filename>`), the standard input (`-d std::cin`) or a random graph on n vertices and, say, $k := 0.5\binom{n}{2}$ average edge count (`-r n k 1`).

Next, invoke `-a a=Chit` to get the chromatic number(s) of the graph(s) input.

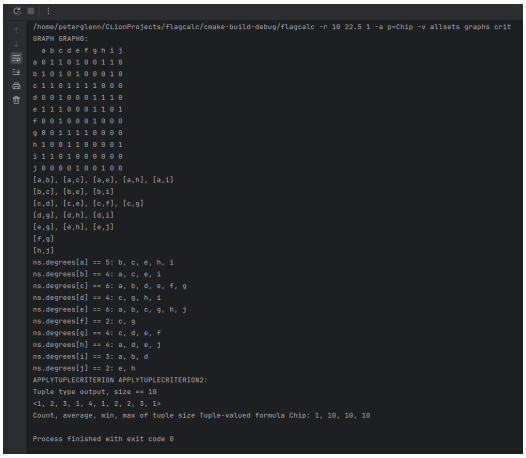
Lastly, choose a "verbosity level" via `-v allsets graphs crit`: we want each set result printed out, we want each graph's adjacency matrix printed out, and we want criteria/measures/tuples/etc. printed out generally. Altogether,

```
-r 10 22.5 1 -a a=Chit -v allsets graphs crit
```



One can also query for the tuple of colors for each vertex:

```
-r 10 22.5 1 -a p=Chip -v allsets graphs crit
```



Finally, one can increase from 1 random graph to l random graphs, and add "allmeas" to verbosity so it outpus the chromatic number of each of the l graphs:

```
-r 8 14 10 -a a=Chit all -v graphs allmeas crit
```

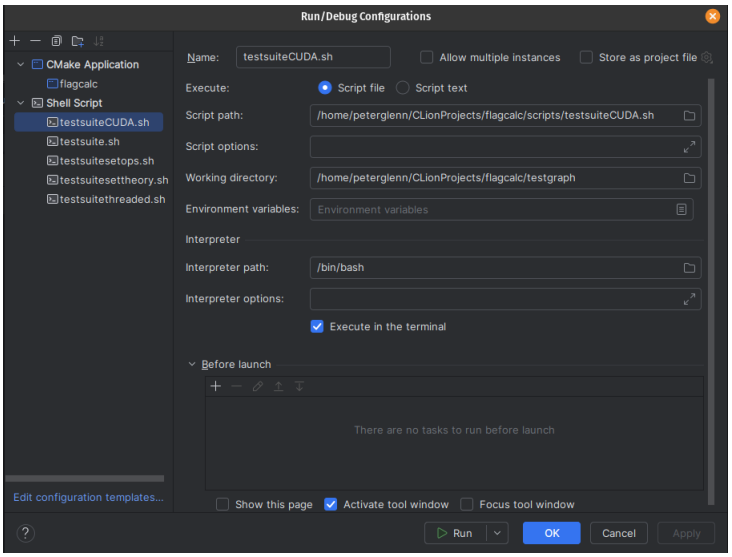
```
ns.degrees[g] == 3: c, d, e
ns.degrees[h] == 4: a, b, c, d
APPLYCRITERION APPLYCRITERION11:
Measure Formula Chit results graph-by-graph:
GRAPH0: 3
GRAPH1: 3
GRAPH2: 4
GRAPH3: 3
GRAPH4: 3
GRAPH5: 4
GRAPH6: 3
GRAPH7: 4
GRAPH8: 4
GRAPH9: 3
Average, min, max of measure Formula Chit: 3.4, 3, 4

Process finished with exit code 0
```

This is by no means the sum of what flagcalc can do; the best user's manual for now is the README.md file (slightly out of date), and simply inspecting visually all the shell scripts in the "scripts" folder, alongside the sample graphs and "stored procedures" in the "testgraph" folder. Also, by now there are several PDFs posted on the author's LinkedIn page, with exercises and instructional material. This LinkedIn URL is:

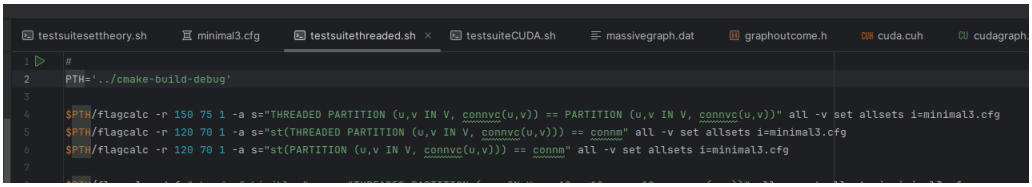
`https://www.linkedin.com/in/peterdglennseminarian`

6. Shell scripts

The heart of flagcalc is the ability to run a long list of queries in batch mode. To do this, go back to "edit configurations" and click the "shell script" drop-down on the left, followed by "Add new run configuration". Use the existing script path as in the following screenshot:



A shell script looks like this:



where the pound symbol is a comment indicator, for giving the script a full name or comment at the beginning; it can be omitted. Scripts should be placed in the "scripts" folder, as indicated in the image above.

7. Specific to Microsoft Windows

Please follow these steps.

(1) Install the JetBrains CLion product on your Windows machine
(2) Install the (free for open-source use) Microsoft Visual Studio C++ tools. Please note the built-in nvcc compiler only works with MSVCC 2019-2022; therefore there is a line in the `CMakeLists.txt` instructing nvcc to use the new (2025) unsupported compiler.
(3) As in the general instructions above, install the CUDA developer's toolkit available from NVIDIA's website
(4) Ensure both the path to `nvcc.exe` and the path to `cl.exe` are in the system's environment variables (click the start menu, then search for "Environment variables"). Both of these paths are long... here they are duplicated as a sample; yours might be different:
    - `C:\Program Files\Microsoft Visual Studio\18\Community\VC\Tools\MSVC\14.50.35717\bin\Hostx64\x64`
    - `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v13.1\bin`

    Note also the path line item `%CLion%`.
(5) Use the Github link `https://github.com/corralledcode/flagcalc.git` in the CLion "New...File from version control" menu. This is now the most up-to-date branch (earlier versions of this document noted the main branch was deprecated by the CUDA-enabled branch; this is no longer the case)
(6) In CLion "Settings/Build,Execution,Deployment/Toolchains" add a Toolchain, click "up" to make it the default, and point its CMake to "MS Visual Studio". Note that in a few moments time it automatically should update all the text boxes with the checked-off compilers, etc.
(7) Click build or add a command line to settings and click Run (one simple command line is just `-h`, which per custom will list all the command-line options)

At this time the Windows implementation has only been tested in a Ubuntu (Pop_OS!) virtual machine through QEMU/KVM, and there it is seen to be runnning several magnitudes more slowly on the basic shell scripts. Whether it runs comparably fast on a native Windows install remains to be seen, however:

If you want to run a shell script (in the project's "Scripts" folder there are six standard ones, or five if you exclude the `testsuiteCUDA.sh`) then follow these steps:

(1) Copy the shell script and rename the copy to have the extension `.bat` rather than `.sh`
(2) find-and-replace all comment delimeters (`#`) with `REM` followed by at least one space.
(3) replace the script's first line, being typically `PTH='../cmake-build-debug'`, with `SET PTH=..\cmake-build-debug`
(4) find-and-replace all occurrences of `$PTH/flagcalc` with `%PTH%\flagcalc`
(5) if there are any multiple-line invocations (continued when the line ends with a backslash), then replace the backslash with a space followed by a caret.

Then, to run the script, be sure to be in the `testgraph` folder (located at `..\testgraph` if one is in the `scripts` folder). Otherwise the script will not find, e.g., `minimal3.cfg` (which specifies some standard "verbosity" levels, i.e. how much info to share to standard output), etc.

8. Further Reading

On page 13 of the document "Intro to Isomorphisms and Automorphisms in Flagcalc" are instructions on "Inputting graphs generally", that describe the language for inputting graphs. For example, you can use

```
./flagcalc -d std::cin -v graphs
```

and when the prompt appears, type in say `abc de END END` for a graph consisting of a K_3 (complete graph on three vertices) alongside a disconnected pair of connected vertices.

The document "Chromatic Number in Flagcalc" has some simple exercises around algorithms for chromatic number (coloring number).

Another document is "Intro to the First-Order Querying Language in Flagcalc". Both of these are found at the author's LinkedIn page,

`https://www.linkedin.com/in/peterdglennseminarian`,

and on the Github page, "main" branch,

`https://github.com/corralledcode/flagcalc/tree/main/documentation`.

Have fun![1]

[1]The author's email address is
`mailto:peter@corralledcode.com`.