Flagmatic User's Guide

Emil R. Vaughan[*]

Version 2.0

January 27, 2013

A tool for researchers in extremal graph theory.

1 Introduction

Flagmatic implements the semi-definite method, part of the flag algebra calculus introduced by Razborov [9]. Let H be an r-graph (or oriented r-graph) on h vertices. Then for a set of r-graphs $\mathscr{F}$, the *Turán H-density* of $\mathscr{F}$ is

$$\pi_H(\mathscr{F}) = \lim_{n\to\infty} \frac{\mathrm{ex}_H(n,\mathscr{F})}{\binom{n}{h}},$$

where $\mathrm{ex}_H(n,\mathscr{F})$ is the maximum number of induced copies of H that an $\mathscr{F}$-free r-graph on n vertices can contain. The graphs in $\mathscr{F}$ are called the *forbidden graphs*.

Flagmatic can solve a variety of types of problem, but the most common use is finding upper bounds on Turán H-densities. Flagmatic (currently) supports 2-graph, 3-graph and oriented 2-graph problems.

The author hopes that over time support will be added for other kinds of problems. In fact Flagmatic 2.0 is a ground-up rewrite of Flagmatic 1.0, written with extensibility in mind. Flagmatic 2.0 is also the first version of Flagmatic that runs entirely within the Sage mathematics system.

Flagmatic is not hard to use, but it is highly recommended that new users spend some time reading this document and studying the examples provided. Problems can typically be solved with 5 commands. For example (the output is not shown):

```
sage: P = ThreeGraphProblem(6, forbid=(4,4), forbid_induced=(4,1))
sage: C = ThreeGraphBlowupConstruction("3:112223331123")
sage: P.set_extremal_construction(C)
sage: P.solve_sdp()
sage: P.make_exact()
```

[*]School of Electronic Engineering and Computer Science, Queen Mary, University of London. Email: e.vaughan@qmul.ac.uk

2 Support

The author is keen to hear reports of any bugs or installation difficulties. Comments or suggestions for improvement are also very welcome! Please contact the developer by email at `e.vaughan@qmul.ac.uk`.

There is a website for Flagmatic:

`http://flagmatic.org/`

3 Installation on Mac OS X

The author has run previous versions of Flagmatic on Linux, but has had trouble getting version 2.0 working correctly. Currently, Mac OS X running on recent (Intel) hardware is the recommended setup for running Flagmatic.

Go to `http://www.sagemath.org/` and download the file

```
sage-5.6-OSX-64bit-10.8-x86_64-Darwin.dmg
```

from one of the download mirrors. At the time of writing, Sage 5.6 is the latest version of Sage, and is the version the author is using. Flagmatic will most probably work on later versions of Sage as well, and may also work on earlier versions of Sage.

Open the file you have downloaded, which is a disk image. Mac OS X will verify the image and then mount it. It will then appear as a new drive in Finder, in the same way that a USB drive appears when it is inserted.

Next, open the Terminal app, and enter the following command to copy Sage to your home directory:

```
    $ cp -a /Volumes/sage-5.6-OSX-64bit-10.8-x86_64-Darwin/sage ~
```

Then enter the following command to set up a symbolic link to allow you to launch Sage easily. This command will prompt you for your password.

```
    $ sudo ln -s ~/sage/sage /usr/local/bin/
```

Once this is done, you will be able to run Sage by entering the `sage` command at the Terminal app's command prompt. If everything has been successful, doing so will result in a message like the following being displayed:

```
$ sage
----------------------------------------------------------------------
| Sage Version 5.6, Release Date: 2013-01-21                         |
| Type "notebook()" for the browser-based notebook interface.        |
| Type "help()" for help.                                            |
----------------------------------------------------------------------
```

Enter `quit` to leave Sage.

To install CSDP, you can download the version the author uses from

http://flagmatic.org/static/bin/csdp

(If you would rather compile it from source, see the Appendix.) The `csdp` file should be moved somewhere where Flagmatic can find it. To do this, enter:

```
$ sudo cp ~/Downloads/csdp /usr/local/bin/
```

Then, to test check it is working:

```
$ csdp
CSDP 6.1.1

Usage:

csdp <input problem> [<final solution>] [<initial solution>]
```

To install Flagmatic, download the zip file. Safari will most likely unzip it automatically; Chrome and Firefox will not. If it is not unzipped, navigate to the Downloads folder in Finder and double-click on the zip file. Then in Terminal type:

```
$ cd Downloads/flagmatic-2.0b2/pkg
$ sage -python setup.py install
```

Next, change up a directory, and run Sage:

```
$ cd ..
$ sage
```

You can now run the examples in the next section.

4 Help and examples

This User's Guide does not attempt to be exhaustive. For more help, it is possible to use Sage's online help system. For example:

```
sage: help(Problem)
sage: help(ThreeGraphFlag.subgraph_density)
```

Flagmatic also comes with many example problems. If you start sage from inside the `flagmatic` directory, it is easy to run an example. For example, here we run the "ff83" example:

```
sage: load examples.sage
sage: E = Example("ff83")
```

One it is run, the `E` object contains the created Problem object as `E.problem`.

Be warned that some of the examples may take a long time to run.

5 Flags

The most basic kind of object used by Flagmatic is the Flag. Flags come in different kinds, the most common being GraphFlag, ThreeGraphFlag and OrientedGraphFlag. We shall look here at GraphFlags, but the other kinds behave similarly.

A GraphFlag is a graph where some of the vertices are marked as being "fixed". The vertices are always called $1,\ldots,n$ for some $n \geq 0$, and the "fixed" vertices are always $1,\ldots,k$ for some $k \geq 0$. To begin with, we shall look at graphs that have no fixed vertices (i.e $k = 0$). A GraphFlag can be created by calling the `GraphFlag` constructor. For example, the following creates a graph with two vertices joined by an edge:

```
sage: G = GraphFlag("2:12")
```

The notation is simply the number of vertices, followed by a colon, followed by a list of edges. The following creates a triangle:

```
sage: G = GraphFlag("3:122331")
```

Specifying an integer to the constructor creates an empty graph. For example the following creates a graph on four vertices with no edges:

```
sage: G = GraphFlag(4)
```

We can then add edges to `G` using the `add_edge` method:

```
sage: G.add_edge((1,2))
sage: G.add_edge((3,4))
sage: G
4:1234
```

Edges can be removed with the `delete_edge` method:

```
sage: G.delete_edge((3,4))
sage: G
4:12
```

GraphFlags can also be created from native Sage graphs, by providing the constructor with a Sage Graph object. For example, we can create the Petersen graph as follows:

```
sage: G=GraphFlag(graphs.PetersenGraph())
sage: G
a:1215162327343845495a6869797a8a
```

Note that the graph is displayed in a slightly odd way: the vertices are represented by 1, ..., 9, a, and the number of vertices by "a". Fortunately, we can display the edges in a more friendly way by using the `edges` property:

```
sage: G.edges
((1, 2), (1, 5), (1, 6), (2, 3), (2, 7), (3, 4), (3, 8), (4, 5), (4, 9),
(5, 10), (6, 8), (6, 9), (7, 9), (7, 10), (8, 10))
```

GraphFlag objects also provide an iterator, which yields the edges:

```
sage: [edge for edge in G]
[(1, 2), (1, 5), (1, 6), (2, 3), (2, 7), (3, 4), (3, 8), (4, 5), (4, 9),
(5, 10), (6, 8), (6, 9), (7, 9), (7, 10), (8, 10)]
```

GraphFlags can be compared with the equality operator, which returns `True` if the graphs are isomorphic. The `is_labelled_isomorphic` method can be used to tell if graphs are *labelled isomorphic*. For example:

```
sage: G = GraphFlag("5:1223344551")
sage: H = GraphFlag(graphs.CycleGraph(5))
sage: G == H
True
sage: G.is_labelled_isomorphic(H)
False
```

Here the second test returned `False`, since Sage's 5-cycle is labelled differently to the one that we created:

```
sage: G
5:1223344551
sage: H
5:1215233445
```

6 Constructions

A construction represents the limit of object of a sequence of extremal constructions

$$(G_n)_{n\in\mathbb{N}}$$

for a problem. Flagmatic works with Construction objects, the most common being instances of the BlowupConstruction class. Two other kinds of Construction are available for 3-graph problems: RandomGraphConstruction and RandomTournamentConstruction. It is also not hard to create new subclasses of Construction.

We can create a Construction representing uniform blow-ups of a 3-cycle as follows:

```
sage: C = GraphBlowupConstruction("3:122331")
```

We can find the (limiting) edge density:

```
sage: C.density()
2/3
```

And the (limiting) triangle density:

```
sage: C.density("3:122331")
2/9
```

Or the 4-vertex induced subgraphs:

```
sage: C.subgraphs(4)
[4:, 4:121314, 4:12132434, 4:1213142324]
```

And the densities of each 4-vertex induced subgraph:

```
sage: C.subgraph_densities(4)
[(4:, 1/27), (4:121314, 8/27), (4:12132434, 2/9), (4:1213142324, 4/9)]
```

7 Problems

Problem objects are central to Flagmatic. Instances of Problem represent Turán-type problems. Problem objects are created by calling one of the functions ThreeGraphProblem, GraphProblem or OrientedGraph-Problem.

For example we can create a Problem that represents Turán's famous problem of maximizing edge density in 3-graphs that do not contain cliques on 4 vertices as follows:

```
sage: P = ThreeGraphProblem(6, forbid=(4,4))
```

Here `forbid=(4,4)` specifies that sets of vertices cannot span 4 edges. In general, forbidding the pair `(k,e)` means that k-sets cannot span e (or more) edges.

The 6 refers to the size of the largest flags considered: the so-called "admissible graphs". Larger values will make the computation harder, but can give better bounds. In general, for 3-graphs, we can always use 6, and sometimes we can use 7.

We can then solve the semi-definite program associated with the program to get a numerical bound:

```
sage: P.solve_sdp()
```

After a short wait, a bound of 0.5616656 will be reported. This bound is not rigorous, as it uses floating point arithmetic. We can get a rigorous bound by typing

```
sage: P.make_exact(2^20)
```

The exact bound reported will vary. I get 1852674441215/3298534883328, which is about 0.56166586. The `2^20` specifies that denominators of 2^{20} should be used in the rounding process. Smaller values will give slightly worse bounds but will produce smaller certificates.

Another problem is that of forbidding the 5-cycle. We can create the 5-cycle as a GraphFlag object, and forbid that:

```
sage: C5 = ThreeGraphFlag("5:123234345451512")
sage: P = ThreeGraphProblem(6, forbid=C5)
```

Or more simply, we can just pass a string:

```
sage: P = ThreeGraphProblem(6, forbid="5:123234345451512")
```

We can also forbid lists of graphs and `(k,e)` pairs:

```
sage: P = ThreeGraphProblem(6, forbid=["5:123234345451512", (4,3)])
```

We can also forbid *induced* subgraphs:

```
sage: P = ThreeGraphProblem(6, forbid=(4,4), forbid_induced=(4,1))
```

By default, the problem is a "maximization" problem: i.e. Flagmatic will find the least upper bound on the edge density possible. In a "minimization" problem Flagmatic will try to find the greatest lower bound possible. For example the following problem is essentially the same as the previous one:

```
sage: P = ThreeGraphProblem(6, forbid_induced=[(4,0), (4,3)], minimize=True)
```

By default all problems concern edge density. We can consider other densities as well, for example the recently-solved problem of determining the maximum density of 5-cycles in a triangle free graph:

```
sage: P = GraphProblem(5, forbid=(3,3), density="5:1223344551")
```

In order to get an exact density result, Flagmatic needs to know about the extremal constructions. In the problem above, the extremal construction is a blowup of a 5-cycle, so we can do:

```
sage: C = GraphBlowupConstruction("5:1223344551")
sage: P.set_extremal_construction(C)
```

Once this is done, we can use the following two commands to solve the SDP and then make the result exact:

```
sage: P.solve_sdp()
sage: P.make_exact()
```

This result differs from the first one we looked at, in that it is tight: the upper and lower density bounds are equal.

8 Appendix: compiling CSDP

Compiling csdp is somewhat tricky, as it needs to be linked with the BLAS and LAPACK linear algebra libraries. The developer of csdp provides binaries on his website. However, it is possible to achieve greater performance on Mac OS X by linking with the Accelerate Framework, and on Linux by linking with the Intel Math Kernel Library.

Some instructions are provided below, but if you are having difficulty compiling csdp, please feel free to contact the author at `e.vaughan@qmul.ac.uk`.

8.1 Mac OS X

The Flagmatic website contains Mac OS X binaries for csdp as well as three versions of sdpa.

To build csdp version 6.1.1 on Mac OS X using the Accelerate Framework, download `Csdp-6.1.1.tgz` and unpack the archive. Change to the `lib` directory and run `make`. Then change to the `solver` directory, and in the `Makefile` change the line

```
LIBS=-L../lib -lsdp -llapack -lblas -lgfortran -lm
```

to

```
LIBS=-L../lib -lsdp -lm -framework Accelerate
```

Then run `make`, and the csdp executable will be created. This can then be moved somewhere where flagmatic will find it.

8.2 Linux

The author suggests that Linux users link csdp with the Intel Math Kernel Library, which is free for non-commercial use, and can be downloaded from

`http://software.intel.com/en-us/articles/non-commercial-software-download/`

To build csdp version 6.1.1 on Linux using the Intel Math Kernel Library, download `Csdp-6.1.1.tgz` and unpack the archive. Change to the `lib` directory and run `make`. Then change to the `solver` directory, and in the `Makefile` change the line

```
LIBS=-L../lib -lsdp -llapack -lblas -lgfortran -lm
```

to

```
LIBS=-L/opt/intel-mkl-10.3/mkl/lib/ia32 -L../lib -lsdp -lmkl_intel -lmkl_sequential \
-lmkl_core -lpthread -lm
```

where `/opt/intel-mkl-10.3/` should be replaced with the directory into which you installed the Intel Math Kernel Library. (You may also need to replace `ia32` with something else if you are using 64-bit Linux.) Then run `make`, and the csdp executable will be created. This can then be moved somewhere where Flagmatic will find it.

You will need to set the `LD_LIBRARY_PATH` environment variable before csdp can run. From the Bash shell you can type

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/intel-mkl-10.3/mkl/lib/ia32
```

Again, change `/opt/intel-mkl-10.3/` to wherever you installed the Intel Math Kernel Library. It is recommended that you put this command in your `.bashrc` file.

9 Redistribution of flagmatic

There are no restrictions on the use of flagmatic, and no license is required to use it.

The developer grants the right of redistribution of the software under the conditions of the following license (based on the 2-clause BSD license). To summarize, *redistribution in source and binary form, with or without modification, is permitted, but credit must be given*.

References

[1] R. Baber and J. Talbot, Hypergraphs do jump, *Combin. Probab. Comput.* **20** (2011), 161–171.
`http://arxiv.org/abs/1004.3733`

[2] D. de Caen, Extension of a theorem of Moon and Moser on complete subgraphs, *Ars Combin.* **16** (1983), 5–10.

[3] V. Falgas-Ravry and E. R. Vaughan, On applications of Razborov's flag algebra calculus to extremal 3-graph theory, preprint (2011).
`http://arxiv.org/abs/1110.1623`

[4] V. Falgas-Ravry and E. R. Vaughan, Turán H-densities for 3-graphs, preprint (2012).
`http://arxiv.org/abs/1201.4326`

[5] A. Grzesik, On the maximum number of C_5's in a triangle-free graph, preprint, 2011.
`http://arxiv.org/abs/1102.0962`

[6] H. Hatami, J. Hladký, D. Král, S. Norine, A. Razborov, On the Number of Pentagons in Triangle-Free Graphs, preprint, 2011.
`http://arxiv.org/abs/1102.1634`

[7] P. Keevash, Hypergraph Turán Problems, *Surveys in Combinatorics 2011*, Springer, 2011.
`http://www.maths.qmul.ac.uk/~keevash/papers/turan-survey.pdf`

[8] D. Mubayi and V. Rödl, On the Turán number of triple systems, *J. Combin. Theory Ser. A* **100** (2002), 136–152.
`http://homepages.math.uic.edu/~mubayi/papers/hypturan.pdf`

[9] A. A. Razborov, Flag algebras, *J. Symbolic Logic* **72** (2007), 1239–1282.
`http://people.cs.uchicago.edu/~razborov/files/flag.pdf` 1

[10] A. A. Razborov, On 3-hypergraphs with forbidden 4-vertex configurations, *SIAM J. Discrete Math.* **24** (2010), 946–963.
`http://people.cs.uchicago.edu/~razborov/files/turan.pdf`