3 Flagmatic package

Emil Vaughan's Flagmatic-2.0 can be found here: `www.flagmatic.org` and the older version (Flagmatic-1.5) here: `http://www.maths.qmul.ac.uk/~ev/flagmatic/`. The latter site contains an overview of results obtained or re-obtained with Flagmatic package).

For updated `setup.py` file to run on top of Sage-6.4.x, clone Flagmatic-2.0 from `https://github.com/jsliacan/flagmatic-2.0.git`. Signatures of several functions that Flagmatic-2.0 uses changed in Sage over time (e.g. method `automorphism_group` in `Graph` class does not take `translation` as argument anymore, see `http://trac.sagemath.org/ticket/14319`) Those changes that were discovered were remedied in Flagmatic-2.0 residing in the referenced Github repository.

There exists an experimental version of Flagmatic, called Flagmatic-dev and you can have it from `https://github.com/jsliacan/flagmatic-dev.git`. This is mainly the version of Flagmatic that is used throughout this text. It operates in three modes (described later in this section):

- Plain mode
- Optimization mode
- Feasibility mode.

3.1 Install and run

To install Flagmatic-dev, for instance, follow the instructions in the `README.md` file (short and brief) or the steps outlined below. Flagmatic-dev and Flagmatic-2.0 is known to work on Mac OS X Yosemite, and Ubuntu 14.04 running Sage-6.4 (as of May 7, 2015).

1. Install Sage-6.4.x. You can download source or binaries from `http://www.sagemath.org/`. Installing from source will take hours.
2. Install CSDP solver.
   ```
   $ sage -i csdp
   ```
3. Download Flagmatic-dev and navigate to its directory
   ```
   $ git clone https://github.com/jsliacan/flagmatic-dev.git
   $ cd flagmatic-dev
   ```
4. Open `install-dev-flagmatic` and change lines 2 and 3 to reflect your directory layout. Save and close. The file looks like this:

Listing 3.1: Contents of `install-flagmatic-dev` file.

```
#!/bin/bash
FLAGMATIC_PKG=/Users/srobik/Github/flagmatic-dev/pkg/
SAGE_SITE_PKGS=/Users/srobik/sage/sage/local/lib/python2.7/site-packages/

# remove old flagmatic versions
cd $SAGE_SITE_PKGS
eval "sudo rm Flagmatic*.egg-info"
eval "sudo rm -rf flagmatic"

# install new flagmatic version
cd $FLAGMATIC_PKG
eval "sudo -E sage -python setup.py install"
```

5. Copy the installation script to `/usr/local/bin` and add permissions to execute.
   `$ cp install-dev-flagmatic /usr/local/bin`
   `$ sudo chmod+x /usr/local/bin/install-dev-flagmatic`

6. Install Flagmatic.
   `$ install-dev-flagmatic`

7. Reload your Terminal (on Mac, `cmd+R` is enough).

8. Run `sage` (assuming it is on your `PATH`). You should see the current version of Sage.
   `$ sage:`

9. Load Flagmatic package.
   `sage:  from flagmatic.all import *`

10. You can now execute Flagmatic commands. Try the following example.

Listing 3.2: Asymptotic version of Mantel's Theorem: $\pi(K_3) = 1/2$.

```
p = GraphProblem(3, forbid="3:121323")
p.solve_sdp(show_output=True)
p.graphs
p.flags
p.types
p?
```

The last line is a standard way to ask for information about a function, variable, class, or any other object. Sage will return the doc string associated with this object.

3.2 Plain mode

For simplicity, we write all expressions asymptotically, i.e. H will stand for its own density in some large graph G. The plain mode is the implementation of the plain flag algebras method. The structure of the problem is the following: minimise (or maximise) a linear combination of subgraph densities in a large graph not containing members of a family $\mathcal{F}$ as induced subgraphs. In other words, let $D^* = a_1 D_1 + \ldots + a_k D_k$ be the quantum graph whose density we intend to maximize – *density quantum graph*, and let $\mathcal{F} = \{F_1, \ldots, F_2\}$. Let $\mathcal{H}_N$ be a family of $\mathcal{F}$-free graphs on N vertices, where N is chosen so that $\max_{1 \leq i \leq k} |V(D_i)| \leq N$. The semidefinite programming problem looks as follows.

$$\begin{aligned} \min \delta : & \\ & D^* + \sum_{\tau \in \mathcal{T}} [\![\mathbf{p}_\tau Q_\tau \mathbf{p}_\tau^T]\!]_\tau \leq \delta \\ & Q_\tau \succeq 0, \quad \forall \tau \in \mathcal{T} \\ & \delta \geq 0 \end{aligned}$$

Of course, all graph densities above are first converted to the linear combinations of graphs in $\mathcal{H}_N$. In short, the plain mode in Flagmatic allows us to ask for an upper bound on $\pi_{D^*}(\mathcal{F})$.

3.2.1 Mantel's theorem

As usually, let us first do the easiest example. Recall that the asymptotic version of Mantel's theorem says that the edge density of a triangle-free graph is at most $1/2$, i.e. $D^* = K_2$ and $\mathcal{F} = \{K_3\}$. In Flagmatic, this looks as below.

Listing 3.3: Bounding asymptotic Turán density of K_3.

```
p = GraphProblem(3, forbid="3:121323")
c = GraphBlowupConstruction("2:12")
p.set_extremal_construction(c)
p.solve_sdp(solver="csdp")
p.make_exact()
p.write_certificate("mantel.cert")
```

3.2.2 Minimizing monochromatic 4-cliques in a 2-colored clique

Let c_n be a edge-2-coloring of a complete graph K_n on n vertices. Let $k_t(c_n)$ be the number of monochromatic complete graphs K_t on t vertices in c_n. Define

$$m_t = \lim_{n \to \infty} \frac{\min_{c_n} k_t(c_n)}{\binom{n}{t}} \tag{3.1}$$

Ramsey's theorem implies that $\frac{\min_{c_n} k_t(c_n)}{\binom{n}{t}}$ is in $[0, 1]$ and it can be shown that it is a non-decreasing sequence in n. Thus the limit in (3.1) exits.

Theorem 2 (Monochromatic K_t's in 2-colored complete graph [Spe12])**.**

$$m_4 \geq \frac{1}{34.7858} = 0.0287473624294971$$

Proof in Flagmatic. Stating the problem in terms of induced densities yields the following

$$m_t = \lim_{n \to \infty} \frac{\text{minex}_{K_4 + \overline{K_4}}(n, \emptyset)}{\binom{n}{t}}$$

where $\text{minex}_{K_4+\overline{K_4}}(n, \emptyset)$ is the minimum, over all n-vertex graphs with no forbidden subgraphs, of the number of 4-sets that induce a K_4 or its complement in that n-vertex graph. In Flagmatic, the code looks as follows.

Listing 3.4: Bounding asymptotic density of monochromatic K_4's in a 2-coloring of a complete graph.

```
p = GraphProblem(8, density=[("4:",1),("4:121314232434", 1)], minimize=True)
p.solve_sdp(solver="csdp")
p.make_exact()
p.write_certificate("monocolor.cert")
```

The above script yields a bound as good as the one in [Spe12]. However, the set of admissible graphs on 8 vertices makes the computations long. A bound of $m_4 \geq 30025/1048576 = 0.02863407135$ can be obtained by running the computations on 7 vertices. The bound is not sharp and the `make_exact` method has no further information to use. Hence rounding changes the bound somewhat. □

3.2.3 Forbidding a tetrahedron and 4-set spanning exactly one 3-edge

The original tetrahedron problem is the Turán problem about forbidding K_4^3, see the statement in Conjecture 1. Many non-isomorphic extremal configurations are known by now – for overview see e.g. [Kee11]. The tetrahedron problem is the smallest from the family of Turán hypergraph problems.

Conjecture 1 (Turán)**.**

$$\pi(K_4) = 5/9$$

Currently, the best bound is due to Razborov [Raz10]. He also pointed out that the extremal construction due to Turán (the corresponding Turán 3-graph) is the only extremal construction from among the known ones which contains no I_4^+ (a 3-graph on four vertices and exactly one 3-edge). Razborov then proved the following result.

Theorem 3 (Razborov [Raz10])**.**

$$\pi(K_4, I_4^+) = 5/9$$

As for the extremal Turán graph, divide the vertices into three parts of equal sizes (as equal as possible). When these parts are arranged in a cyclic order, put in all edges of the following two types. The first type has one vertex in each part. The second type has two vertices in one part, and the remaining vertex in the next part in clockwise direction. Clearly, the edge count in this extremal construction is $3 \cdot \left[\frac{n}{3} \cdot \binom{n/3}{2}\right] + \left(\frac{n}{3}\right)^3 = \frac{5n^3-6n^2}{54} = \frac{5}{54}n^3 - O(n^2)$. So we know about the normalized Turán numbers that $\pi(n, K_4) \geq 5/9 - O(1/n)$. Hence $\pi(K_4) \geq 5/9$.

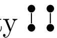
The Flagmatic code that proves the asymptotic upper bound of 5/9 is below.

Listing 3.5: Bounding asymptotic Turán density of $\{K_4^3, I_4^+\}$.

```
p = ThreeGraphProblem(7, forbid_induced=["4:123124134234", "4:123"])
c = ThreeGraphBlowupConstruction("3:112223331123")
p.set_extremal_construction(c)
p.solve_sdp(solver="csdp")
p.make_exact()
p.write_certificate("not-tetrahedron.cert")
```

3.3 Optimization mode

As in the sections above, let n be the order of admissible graphs in $\mathcal{H}_n$. Assume we want to maximise a linear combination of graphs, say $D^* = a_1D_1 + \ldots a_tD_t$. Assume further that we want to maximise D^* subject to constraints, each of which can be expressed as a linear inequality of graph densities (i.e. *quantum graph*). We call each such inequality an *assumption*. In other words, let $G = \sum_{i=1}^{q} d_iG_i$ be a quantum graph and $d \in \mathbb{R}$. Then $G \geq d$ is an assumption.

For instance, imagine that we want to maximize the number of triangles in a graph that has a density of 4-vertex sets spanning exactly two edges at least d. That is, we want $\max K_3$ such that  $= K_2 \cup K_2 \geq d$ for some fixed d. Here $H^* = K_3$ and the assumption in this case is the inequality  $\geq d$. Let $\mathcal{H}_N$ be the family of N-vertex admissible graphs – they are like a currency, all computations are done in terms of their densities. Finally, let $\mathcal{T}$ be the set of all types used in flag algebra computations. Then in general, the assumption is of the form

$$S = \sum_{\substack{W \in \mathcal{F}' \subseteq \mathcal{F}^\sigma \\ |\mathcal{F}'| < \infty}} b_W W \geq b, \quad b_w \in \mathbb{R}$$

Let $l = \min_{W \in \mathcal{F}'} \{N - |V(W)|\}$ (N is always chosen so that it is not smaller than any of the graphs in D^* or assumptions). Let also $M = |\mathcal{F}'|$. Then the problem description looks like this:

$$\begin{aligned}
&\min \delta : \\
&\quad D^* + \left[\!\!\left[(S_1 - b_1) \sum_{i=1}^{l_1} c_i^1 F_i^1 \right]\!\!\right]_{\sigma_1} + \ldots + \left[\!\!\left[(S_M - b_M) \sum_{i=1}^{l_M} c_i^M F_i^M \right]\!\!\right]_{\sigma_M} + \sum_{\tau \in \mathcal{T}} \left[\!\left[\mathbf{p}_\tau Q_\tau \mathbf{p}_\tau^T \right]\!\right]_\tau \leq \delta \\
&\quad Q_\tau \succeq 0, \quad \forall \tau \in \mathcal{T} \\
&\quad c_i^1 \geq 0, \quad \forall i = 1, \ldots, l_1 \\
&\quad \vdots \\
&\quad c_i^M \geq 0, \quad \forall i = 1, \ldots, l_M \\
&\quad \delta \geq 0
\end{aligned}$$

In Flagmatic-dev, you can express your wish to use optimization mode by passing the argument `mode="optimization"` to the Problem class. Doing so will allow you to use the `add_assumption` method of the Problem class. See the examples below for demonstration of usage.

Assume that the quantum assumption graph is a linear combination of k simple graph flags. Then the signature of `add_assumption` is

`add_assumption(`τ`,[(`G_1`,`c_1`),...,(`G_1`,`c_1`)],`d`, equality=False)`

- τ is the graph-string representing the type-graph. The quantum assumption graph is a linear combination of τ-flags – the restriction is that all graphs in the quantum assumption graph must be over same typegraph (their labelled parts are labelled-isomorphic).
- `[(`G_1`,`c_1`),...,(`G_k`,`c_k`)]` is the quantum assumption graph, a linear combination of τ-flags (not necessarily of the same order).
- d is the RHS of the assumption which if of the form $G = d$ or $G \geq d$ for some $d \in [0, 1]$ and G a quantum graph of τ-flags. It is permissible to enter d as a fraction, i.e. `1/8` is correct syntax. Otherwise use decimal form, e.g. `0.125`.
- `equality=False` is an argument which specifies whether the assumption is an equality $G = d$ or an inequality $G \geq d$. The latter is default, so equality needs to be explicitly specified.

The user is allowed to enter as many assumptions as she requires by repeatedly calling the `add_assumption` method.

3.3.1 Mantel's theorem revisited

Notice the following fact: forbidding a graph F and requiring that $F = 0$ are asymptotically the same constraints, despite one of them being exact and the other one asymptotic. We can see this on an example of Mantel's theorem. In Section 3.2 we used the following code to obtain asymptotic version of Mantel's theorem. Flagmatic's answer follows.

Listing 3.6: Bounding asymptotic Turán density of K_3. (plain mode)

```
p = GraphProblem(3, forbid="3:121323")
c = GraphBlowupConstruction("2:12")
p.set_extremal_construction(c)
p.solve_sdp(solver="csdp")
p.make_exact()
```

Listing 3.7: Output

```
Forbidding 3:121323 as a subgraph.
Generating graphs...
Generated 3 graphs.
Generating types and flags...
Generated 1 types of order 1, with [2] flags of order 2.
Computing products.
Writing SDP input file...
Running SDP solver...
Returncode is 0. Objective value is 0.50000001.
Checking numerical bound...
Bound of 1/2 appears to have been met.
The following 2 graphs appear to be sharp:
0.499999996063 : graph 0 (3:)
0.500000006567 : graph 2 (3:1213)
Type 0 (2 flags) blocks: [2]
Creating bases.
Transforming matrices.
Rounding matrices.
Constructing R matrix.
Constructing DR matrix.
DR matrix has rank 1.
All eigenvalues appear to be positive.
Bound of 1/2 attained by:
1/2 : graph 0 (3:)
1/2 : graph 2 (3:1213)
Diagonalizing.
Verifying.
```

Specifying the △-freeness through assumptions is demonstrated in the following flagmatic script.

Listing 3.8: Bounding asymptotic Turán density of K_3. (optimization mode)

```
p = GraphProblem(3, mode="optimization")
p.add_assumption("0:", [("3:121323(0)", 1)], 0, equality=True)
c = GraphBlowupConstruction("2:12")
p.set_extremal_construction(c)
p.solve_sdp(solver="csdp")
p.make_exact()
```

Listing 3.9: Output

```
Generating graphs...
Generated 4 graphs.
Generating types and flags...
Generated 1 types of order 1, with [2] flags of order 2.
Computing products.
Added 1 quantum graphs.
Added 1 quantum graphs.
Determining which graphs appear in construction...
```

```
Density of construction is 1/2.
Found 1 zero eigenvectors for type 0.
Writing SDP input file...
Running SDP solver...
Returncode is 0. Objective value is 0.5.
Checking numerical bound...
Bound of 1/2 appears to have been met.
The following 2 graphs appear to be sharp:
0.499999999032 : graph 0 (3:)
0.500000001613 : graph 2 (3:1213)
Type 0 (2 flags) blocks: [2]
Creating bases.
Transforming matrices.
Rounding matrices.
Constructing R matrix.
Constructing DR matrix.
DR matrix (density part) has rank 1.
DR matrix has rank 2.
All density coefficients are non-negative.
All eigenvalues appear to be positive.
Bound of 1/2 attained by:
1/2 : graph 0 (3:)
1/2 : graph 2 (3:1213)
Diagonalizing.
Verifying.
```

Notice how assumptions are specified. First, in line 1 of Listing 3.13, we set `mode` to `"optimization"`. This will allow us to use the method `add_assumption` in line 2 of the same script sample. In our case, we don't want our quantum assumption graph () to be labelled. So the typegraph τ is on 0 vertices and has no edges (see line 2 of 3.13, `"0:"`). As already mentioned, the quantum assumption graph is just $1 \cdot$ . We want the asymptotic density of  to be 0, so $d = 0$. Also, we want it to be equal to 0, not greater of equal. So `equality=True` needs to be set.

3.3.2 Modification of Mantel's theorem

Let us consider the following example. We would like to maximize the number of edges in a -free graph, in which additionally we require the density of  to be at most 2/3. As expected, this forces the edge density to decrease.

Listing 3.10: Bounding asymptotic Turán density of K_3 given that $\overline{P_3} \leq 2/3$.

```
p = GraphProblem(3, forbid="3:121323", mode="optimization")
p.add_assumption("0:", [("3:1223", -1)], -2/3)
p.solve_sdp(solver="csdp")
```

Listing 3.11: Output

```
Forbidding 3:121323 as a subgraph.
Generating graphs...
Generated 3 graphs.
Generating types and flags...
Generated 1 types of order 1, with [2] flags of order 2.
Computing products.
Added 1 quantum graphs.
Writing SDP input file...
Running SDP solver...
Returncode is 0. Objective value is 0.47140453.
Checking numerical bound...
```

3.3.3 Sós problem

Listing 3.12: Sós problem

```
N = binomial(4,2)

def dens(pp, n, k):
    return binomial(n,k)*pp^k*(1-pp)^(n-k)

sp = GraphProblem(4,
                  density=[("4:12132434(0)", -4), ("4:12233124(0)", 1),
                           ("4:1434(0)", 1), ("4:1324(0)", -4)],
                  types=["2:","2:12"],
                  mode="optimization")

sp.add_assumption("0:", [("4:(0)", 1)], dens(1/2, N, 0), equality=True)
sp.add_assumption("0:", [("4:12(0)", 1)], dens(1/2, N, 1) , equality=True)
sp.add_assumption("0:", [("4:1223(0)", 1), ("4:1234(0)", 1)], dens(1/2, N, 2),
                  equality=True)
sp.add_assumption("0:", [("4:121314(0)", 1), ("4:122334(0)", 1), ("4:122331(0)", 1)],
                  dens(1/2, N, 3), equality=True)
sp.add_assumption("0:", [("4:12233441(0)", 1), ("4:12233134(0)", 1)], dens(1/2, N, 4),
                  equality=True)
sp.add_assumption("0:", [("4:1223344113", 1)], dens(1/2, N, 5), equality=True)
sp.add_assumption("0:", [("4:122334411324", 1)], dens(1/2, N, 6), equality=True)
sp.solve_sdp(solver="csdp")
```

Listing 3.13: Output

```
Generating graphs...
Generated 11 graphs.
Generating types and flags...
Generated 0 types of order 0, with [] flags of order 2.
Generated 2 types of order 2, with [4, 4] flags of order 3.
Computing products..
Added 1 quantum graphs.
Added 1 quantum graphs.
Added 1 quantum graphs.
Added 1 quantum graphs.
Added 1 quantum graphs.
Added 1 quantum graphs.
Added 1 quantum graphs.
Added 1 quantum graphs.
Added 1 quantum graphs.
Added 1 quantum graphs.
Added 1 quantum graphs.
Added 1 quantum graphs.
Added 1 quantum graphs.
Added 1 quantum graphs.
Writing SDP input file...
Running SDP solver...
Returncode is 0. Objective value is 2.6474621e-10.
Checking numerical bound...
```