MACAULAY2 - RTG SEMINAR

SEPTEMBER 11TH, 2023

You can access Sage online at https://sagecell.sagemath.org/ and run the code below there.

It is also possible to download Sage (without root access) on the department machines, using the instructions at https://doc.sagemath.org/html/en/installation/conda.html#sec-installation-conda. You can then run it from the terminal and get the interpreter-like interface.

It is also straightforward to download it on Mac (with or without admin privileges).

If you know Python, you know Sage (in quite a literal sense). Sage also has an interface to use Macaulay2, which is what we will be looking at.

```
G = Graph([(0, 1), (1, 2), (0, 2)])
G.plot().show()
V = G.vertices()
print(V)
E = G.edges(labels=False)
print(E)
```

If you are using the Sage cell, it looks like you can simply type the variable name and it will print it – if that is the last line of the code. Try changing the `print(E)` to just `E` in the last line.

So far, I have only used the Graph library by Sage. Now, let us use the Macaulay2 interface. The documentation can be found at https://doc.sagemath.org/html/en/reference/interfaces/sage/interfaces/macaulay2.html.

```
R = macaulay2('ZZ/2[x_0, x_1, x_2]')
print(R)
x = R.gens()
print(x)
```

The outputs should make sense. Note that `R` and `x` are actually Sage variables. On the other hand,

```
print(x_0)
```

will give an error. However, we can access this using the list `x` that we created earlier. So, something like

```
f = x[0] - x[1] * x[2]
```

gives a valid element that you can try printing.[1]

```
mons = [x[v] * x[w] for v, w in E]
```

Try printing the above to see we have created. The syntax comes from Python.

[1] `x` is a list whose elements can be accessed as `x[i]`, where indexing starts from 0. In our example, since our variables are also indexed from 0, things line up as expected.

A natural (monomial) ideal associated to a graph is the edge ideal. What we have created above is the (canonical monomial) minimal generating set of the ideal. The above is a list in Sage, whose elements are Macaulay2 objects[2]. We now turn this into an actual ideal.

```
I = macaulay2.ideal(mons)
C = I.res()
B = I.res().betti()
print(C)
print(B)
print(C.dot('dd'))
```

This can be useful! For example, putting together the above things, we have a function below that gives the Betti table of the edge ideal of a graph (computed over `ZZ/2`).[3]

```
def bettiTable(G):
    V = G.vertices(sort=True)
    E = G.edges(labels=False)
    vars = ','.join([f'x_{v}' for v in V])
    R = macaulay2(f'ZZ/2[{vars}]')
    x = R.gens()
    I = macaulay2.ideal([x[v] * x[w] for v, w in E])
    return I.res().betti()
```

With a little more fussing about how M2 stores Betti tables as hash tables, one could make a function that gives the total betti numbers. (I am not sure if there's an existing M2 command that gives it directly.) Here's a code that does that. For the time being, let us blindly use it.

```
def totalBetti(G):
    B = bettiTable(G)
    B = dict(B)
    tot = {}
    for key in B:
        i = key[0].sage()
        if i not in tot:
            tot[i] = 0
        tot[i] += B[key]
    return [tot[i] for i in range(len(tot))]
```

Now, we could, for example, look at the total Betti numbers of complete graphs.

```
for n in range(1, 10):
    print(totalBetti(graphs.CompleteGraph(n)))
```

In the above, we are making use of the `graphs` library of Sage. There are many other classes of graphs that can be called by a command. There are also ways of going through all graphs with n vertices (and more restrictions can be put on number of edges, connectedness, if it's claw-free, et cetera).

Getting data like this can be useful for conjecturing things: For example, for each i, entering $\beta_i(I(K_n))$ for few values of n, in OEIS, shows that these are binomial coefficients.

[2] I think technically they are still Sage objects that wrap a Macaulay2 object.
[3] The function assumes that `G` has vertex set of the form $\{0, \ldots, n-1\}$.

Another example of a Sage function I had used in the past to do things with ideals: I had wanted to generate all (nonzero) homogeneous quadratic ideals in $\mathbb{F}_2[x, y, z]$. There are 63 monomials and thus, $2^{63} - 1 > 9 \cdot 10^{18}$ possible generating sets. However, there are many redundancies: $(x^2, y^2) = (x^2, x^2 + y^2)$ for example.
In fact, these ideals are in natural bijective correspondence with nonzero subspaces of $\mathbb{F}_2\{x^2, y^2, z^2, xy, yz, xz\}$. This brings the number down to 2824. Moreover, Sage has a method to generate all the subspaces of $\mathbb{F}_2^6$, which let me get all the ideals.