GraphExamples

Example session showing Sage's graph theory capabilities

Stefan van Zwam

In this notebook I list some of the ways in which graphs can be constructed, inspected, and manipulated, with a view towards mimicking these capabilities for matroids.

0. Sage

Sage can do much that your favorite computer algebra system has to offer.

\[
\begin{align*}
2+2 &= 4 \\
x &= \text{var('x')} \\
\text{integrate}( x^3 + 3/4,x) &= \frac{1}{4} x^4 + \frac{3}{4} x \\
f(x) &= x^3 + 3/4 \\
f.plot((x,0,3))
\end{align*}
\]
a = var('a')
F.<a> = GF(4)
F

\[ F_{2^2} \]

\[
\begin{pmatrix}
0 & 0 & 0 & 0 \\
0 & a+1 & 1 & a \\
0 & 1 & a & a+1 \\
0 & a & a+1 & 1
\end{pmatrix}
\]

(Note: I marked the "typeset" checkbox at the top to get the output to look so nice. Everything below was evaluated with the checkbox unchecked.)

For a more thorough introduction, Sage's tutorial is a good place to start. Some hints:

- <shift>+<enter> or clicking the "evaluate" link will evaluate a cell
- Hitting <tab> invokes autocompletion
- Most things you create are "objects" in the programming language sense. This means that there are functions associated with them. If you type the name of the object, followed by a period, and hit <tab>, you see a list of all functions it knows of.
- Help can be obtained by typing a question mark after an expression and hitting either <tab> or <shift>+<enter>

1. Where to find graphs
On to graph theory! A graph can be constructed in various ways:

```python
# From a built-in constructor (note: the # symbol denotes a
# comment and is not evaluated)

petgraph = graphs.PetersenGraph()
petgraph.show()

Hint: try typing "graphs.<tab>" to see all constructors. Try typing "petgraph.<tab>" to see all
methods associated with a graph object. And use "graphs?" to bring up the documentation.

petgraph.cliques_maximal()

    [[0, 1], [0, 4], [0, 5], [2, 1], [2, 3], [2, 7], [3, 4], [3, 8]
     [1], [6, 8], [6, 9], [7, 5], [7, 9], [8, 5], [9, 4]]

# With parameters

graphs.CompleteMultipartiteGraph([1,2,1,2]).show()
```
Next, we extract some graphs from Sage's built-in SQL database of graphs with up to 7 vertices. This database contains quite some information on these graphs. Bigger databases are available for download.

```python
# From a database

gdb = GraphDatabase()
gdb.db_info(tablename='graph_data')
['complement_graph6', 'eulerian', 'graph6', 'lovasz_number',
 'num_cycles', 'num_edges', 'num_hamiltonian_cycles', 'num_vertices',
 'perfect', 'planar']

# the nonplanar graphs on 6 vertices and 10 elements

nonplanargraphs6_10 = GraphQuery(gdb, display_cols = ['graph6',
 'num_vertices', 'num_edges', 'degree_sequence'], num_vertices =
 ['=', 6], num_edges = ['<=', 10], planar = ['=', 0])
for gr in nonplanargraphs6_10.get_graphs_list():
    gr.show()
```
Of course, one can input a graph directly, in a variety of ways:

```
# From an adjacency matrix
Gadj = Graph(Matrix(
    [[0,1,1,0],
     [1,0,1,1],
     [1,1,0,1],
     [0,1,1,0]]
))
Gadj.show()

type(Gadj)

<class 'sage.graphs.graph.Graph'>
```

```
# From an incidence matrix
Ginc = Graph(Matrix(
    [[-1,-1,0,0,0],
     [1,0,-1,-1,0],
     [0,1,1,0,-1],
     [0,0,0,1,1]]
))
Ginc.show()
```
2. What to do with them
Next, we consider some of the methods of the Graph object. Again, after defining a graph, you can use tab completion to get a list of its methods. Try typing Ginc.*TAB*

```
Ginc.automorphism_group()
```

```
Permutation Group with generators [(3,4), (1,2)]
```

```
Ginc.incidence_matrix()
```

```
[-1 -1  0  0  0]
[ 0  1 -1 -1  0]
[ 1  0  0  1 -1]
[ 0  0  1  0  1]
```

```
Ginc.vertex_connectivity()
```

```
2.0
```

```
# This returns a minimum-weight edge cut in addition to the number
Ginc.edge_connectivity(value_only = False)
```

```
[2.0, [(1, 3, None), (2, 3, None)]
```

```
Ginc.is_isomorphic(Glist)
```

```
False
```

```
# Warning: every variable in Sage is a reference! So the command below makes Glist2 a reference to the same object that Glist refers to!
Glist2 = Glist
Glist2.show()
```
# To make a new object out of the old one, use copy()

Glist3 = copy(Glist)
Glist3.show()

# Edge deletion happens inside the object, rather than returning a new one.

Glist.delete_edge((3,4,'f'))
Glist.show()
# Note that Glist2 has changed as well, but Glist3 hasn't.

Glist2.show()

Glist3.show()

# Now the two graphs Ginc and Glist are isomorphic

Ginc.is_isomorphic(Glist)

True
# Finally: minor testing (it uses some MIP formulation internally)

```python
petgraph.minor(Ginc)
```

```python
{0: [0], 1: [1], 2: [2, 4, 7, 9], 3: [3, 5, 6, 8]}
```