Spicing Up Your Research in Combinatorics with Sage

February 12, 2011

Rob Beezer
University of Puget Sound

14th Coast Combinatorics Conference
University of Victoria

1 Disclaimers

- I may be the wrong person to give this talk!
- Open source fanatic
- Member, Sage developer community
- Sage development: linear algebra, graph theory, group theory
- But I will *try* to be objective

2 Introduction

- Sage: Open source software for mathematics
- Mission: "Creating a viable free open source alternative to Magma, Maple, Mathematica and Matlab."

- Consolidates over 100 packages, such as:
  - NetworkX — Graph Theory
  - GAP — Groups, Algorithms, Programming
  - ATLAS — Automatically Tuned Linear Algebra System
  - IML — Integer Matrix Library
- Command Line, Batch, Notebook Interface
- FREE!

3 Graph Theory

Create graphs in a natural way:

```
harary = Graph([(0,1), (1,2), (2,3), (3,0), (1,3)])

harary

harary.plot()

harary.num_verts(), harary.num_edges()

harary.is_planar()

H=harary.hamiltonian_cycle()
H.plot()

harary.degree_sequence()

sorted(harary.degree_sequence())
```

3.1 Graph Generators

Many pre-defined graphs (digraphs, too):

```
graphs.
```

```
trees_iterator = graphs.trees(8)
T8 = list(trees_iterator)
T8
```

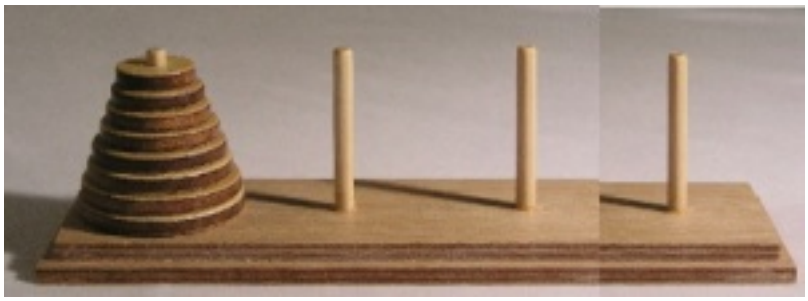
From a path to a star:

```
[tree.diameter() for tree in T8]
```

Visually:

```
graphs_list.show_graphs(T8)
```

3.2 Tower of Hanoi Graph



- `graphs.HanoiTowerGraph(n, d)`
- Generalize to n pegs and d disks
- State graph: intermediate configurations, edges are "one move"
- Labels: d-tuple, large disk to small disk; entries are peg numbers
- Example: $n = 3$, $d = 4$: $(2, 0, 2, 1)$

```
T = graphs.HanoiTowerGraph(3, 4, positions=True)
T.show(figsize=12)
```

A solution is path between states "all the disks on one peg" and "all the disks on another peg."

```
solution=T.shortest_path((0,0,0,0), (1,1,1,1))
solution
```

Minimum number of moves:

```
len(solution) - 1
```

```
T.diameter()
```

More general:

```
T = graphs.HanoiTowerGraph(4, 3, positions=True)
T.show(figsize=12)
```

```
T = graphs.HanoiTowerGraph(4, 4, labels=False, positions=True)
T.show(figsize=12)
```

Forget about graphics, work with graph itself.

```
T = graphs.HanoiTowerGraph(4, 8, labels=False, positions=False)
T.num_verts()
```

Code vertices to integers: d-tuples, base n. All disks on peg 0, move to all disks on peg 3.

```
solution = T.shortest_path(0, 4^8-1)
solution
```

```
len(solution)-1
```

Theorem: automorphisms of the state graph are just the obvious ones (renumber pegs)

```
T = graphs.HanoiTowerGraph(4, 6, labels=False, positions=False)
A = T.automorphism_group()
A.order()
```

```
S4 = SymmetricGroup(4)
S4.is_isomorphic(A)
```

Automorphisms are computed via Brendan McKay's `nauty` algorithm, reimplemented as `NICE`.

3.3 Online Help

Tab-completion, online syntax and tested examples (?), source code (??).

```
G = graphs.GrotzschGraph()

G.degree_seq
```

3.4 Algebraic Graph Theory

Comprehensive support for groups via GAP. If need be, can ship *any* GAP command directly from Sage to GAP.

```
P = graphs.PetersenGraph()
A = P.automorphism_group()
A

A.order()

A.is_transitive()

S = A.stabilizer(9); S

sg = S.subgroups(); sg

[H.order() for H in sg]

H.is_isomorphic(DihedralGroup(6))

ds = A.derived_series()

[K.order() for K in ds]

B = ds[1]
B.is_isomorphic(AlternatingGroup(5))
```

Comprehensive support for linear algebra.

```
A = random_matrix(ZZ, 1000, 1000)
time A.determinant()

K = graphs.KneserGraph(8,3)
```

```
adj = K.adjacency_matrix()
adj

K.spectrum()
```

A small "singular graph." (I. Sciriha, 2007)

```
S = graphs.CycleGraph(4)
S.add_vertices([4, 5, 6])
S.add_edges([(2,4), (2,5), (2,6)])
S.add_edges([(3,4), (3,5), (3,6)])
S.plot()

graph_editor(S)

adj = S.adjacency_matrix()
ker = adj.kernel()
ker
```

Notice this is the kernel over the integers, and is computed as a module. It is easy to upgrade to the rationals.

```
adjQ = adj.change_ring(QQ)
kerQ = adjQ.kernel()
kerQ
```

A vector space in Sage behaves like a vector space, it has dimension, you can interect two of them, ...

```
kerQ.
```

4 Combinatorics

MuPAD-Combinat was sold to MathWorks (i.e. Matlab) in September 2008. With foresight, MuPAD contributors had switched to Sage in June 2008. So there is an active and experienced group contributing combinatorics to Sage as "sage-combinat."
The usual functions:

```
stirling_number2(500,30)
```

```
number_of_derangements(["a", 'b', 'c', 'd', 'e'])
```

The usual objects (note multisets):

```
permutations([0,1,1,2,2,2])

combinations([0,1,1,2,2,2], 3)

derangements(["a", 'b', 'c', 'd', 'e'])
```

Ranking and Unranking (uses classes):
(preserve order for `.rank()`)

```
C = Combinations(["cat", "dog", "pig", "fox", "bug", "rat"], 3)

C.unrank(15)

C.rank(['cat', 'dog', 'bug'])
```

Polynomials:
A symbolic object:

```
var('t')
p = bernoulli_polynomial(t, 5); p

diff(p, t)

p.parent()
```

A function:

```
var('t')
q(t) = bernoulli_polynomial(t, 5); q

q(10)

q.parent()
```

A polynomial over the rationals, i.e. in $\mathbf{Q}[t]$:

```
t = polygen(QQ, 't')
r = bernoulli_polynomial(t, 5); r
```

```
r.is_irreducible()

r.factor()

r.parent()
```

Lots of esoterica:

```
S = SkewPartitions(4)
s = S[25]; s

print s.diagram()
```

Much, much more: crystals, posets, root systems, symmetric functions, words.

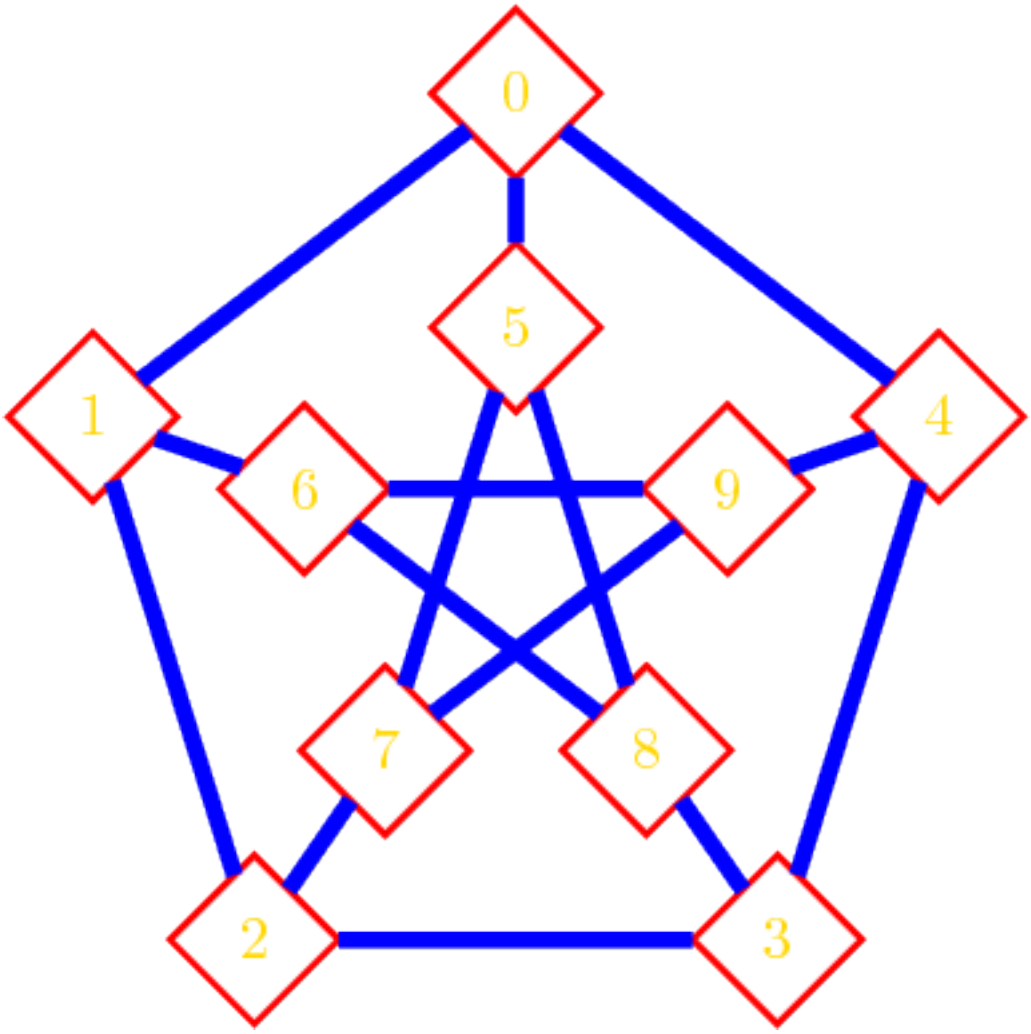
5 Goodies

Excellent LaTeXsupport.

```
latex(integrate(sec(x), x))

A = random_matrix(QQ, 4, num_bound=9, den_bound=9)
latex(A)
```

LaTeX versions of graphs:

```
P = graphs.PetersenGraph()
P.set_latex_options(vertex_shape='diamond', vertex_color='red', vertex_label_c

latex(P)

from sage.graphs.graph_latex import setup_latex_preamble
setup_latex_preamble()
latex.jsmath_avoid_list('tikzpicture')
```



Embedded Word Processor (Shift-Click) is TEX-aware.

SageTEX allows you to call Sage to compute values, expressions or plots for automatic inclusion in your LATEX document.

• Input: `Today's date \$2011-02-12=20110212\$ factors as \sage{factor(20110212)}.`

• Output: Today's date $2011-02-12 = 20110212$ factors as $2^2 \cdot 3^2 \cdot 173 \cdot 3229$.

Can author in LATEX and produce worksheets (this is an example).

6 Interacts and Python Extensions

• Full support for any Python package.
• Easy-to-construct interactive demonstrations.
Code adapted from a demonstration by William Stein

```
import pylab
A_image = pylab.mean(pylab.imread(DATA + 'mystery.png'), 2)
@interact
def svd_image(i=(1,(1..50)), display_axes=True):
    u,s,v = pylab.linalg.svd(A_image)
    A     = sum(s[j]*pylab.outer(u[0:,j], v[j,0:]) for j in range(i))
    show(matrix_plot(A), axes=display_axes, figsize=(11,5))
    html('<h2>Compressed using %s singular values</h2>'%i)
```

Code from Sage Wiki (Interacts) by Pablo Angulo.

```
%hide
def animate_contraction(g, e, frames = 12, **kwds):
    v1, v2 = e
    if not g.has_edge(v1,v2):
        raise ValueError, "Given edge not found on Graph"
    ls = []
    posd = g.get_pos()
    for j in range(frames):
        gp = Graph(g)
        posdp = dict(posd)
        p1 = posdp[v1]
        p2 = posdp[v2]
        posdp[v2] = [a*(frames-j)/frames + b*j/frames
                     for a,b in zip(p2,p1)]

        gp.set_pos(posdp)
        ls.append(plot(gp, **kwds))
    return ls

def animate_vertex_deletion(g, v, frames = 12, **kwds):
    kwds2 = dict(kwds)
    if 'vertex_colors' in kwds:
```

```
            cs = dict(kwds['vertex_colors'])
            for c, vs in kwds['vertex_colors'].items():
                if v in vs:
                    vs2 = list(vs)
                    vs2.remove(v)
                    cs[c] = vs2
            kwds2['vertex_colors'] = cs
        else:
            kwds2 = dict(kwds)
        g2 = Graph(g)
        posd = dict(g.get_pos())
        del posd[v]
        g2.delete_vertex(v)
        g2.set_pos(posd)
        return [plot(g, **kwds),plot(g2, **kwds2)]*int(frames/2)

    def animate_edge_deletion(g, e, frames = 12, **kwds):
        v1, v2 = e
        g2 = Graph(g)
        g2.delete_edge(e)
        return [plot(g, **kwds),plot(g2, **kwds)]*int(frames/2)

    def animate_glide(g, pos1, pos2, frames = 12, **kwds):
        ls = []
        for j in range(frames):
            gp = Graph(g)
            pos = {}
            for v in gp.vertices():
                p1 = pos1[v]
                p2 = pos2[v]
                pos[v] = [b*j/frames + a*(frames-j)/frames
                            for a,b in zip(p1,p2)]
            gp.set_pos(pos)
            ls.append(plot(gp, **kwds))
        return ls

    def medio(p1, p2):
        return tuple((a+b)/2 for a,b in zip(p1, p2))
```

```
def new_color():
    return (0.1+0.8*random(), 0.1+0.8*random(), 0.1+0.8*random())

def animate_minor(g, m, frames = 12, pause = 50, step_time = 100):
    '''Crea una animacin que muestra cmo un grafo tiene un menor m
    '''
    posd = dict(g.get_pos())
    posg = posd.values()
    posm = m.get_pos().values()
    xmax = max(max(x for x,y in posg), max(x for x,y in posm))
    ymax = max(max(y for x,y in posg), max(y for x,y in posm))
    xmin = min(min(x for x,y in posg), min(x for x,y in posm))
    ymin = min(min(y for x,y in posg), min(y for x,y in posm))
    dd = g.minor(m)

    #Set colors
    m_colors = dict((v,new_color()) for v in m.vertices())
    g_colors = dict((m_colors[k],vs)
                      for k,vs in dd.items())

    extra_vs = (set(g.vertices()) -
                set(v for vs in dd.values()
                      for v in vs))
    g_colors[(0,0,0)] = list(extra_vs)

    #pics contains the frames of the animation
    #no colors at the beggining
    gg = Graph(g)
    pics = [plot(gg)]*frames

    #First: eliminate extra vertices
    for v in extra_vs:
        pics.extend(animate_vertex_deletion(gg, v, frames,
                          vertex_colors = g_colors))
        gg.delete_vertex(v)
        del posd[v]
        g_colors[(0,0,0)].remove(v)
```

```
del g_colors[(0,0,0)]

#Second: contract edges
for color, vs in g_colors.items():
    while len(vs)>1:
        for j in xrange(1,len(vs)):
            if gg.has_edge(vs[0], vs[j]):
                break
        pics.extend(animate_contraction(gg, (vs[0], vs[j]), frames,
                            vertex_colors = g_colors))
        for v in gg.neighbors(vs[j]):
            gg.add_edge(vs[0],v)
        gg.delete_vertex(vs[j])
        del posd[vs[j]]
        gg.set_pos(posd)
        posd = dict(gg.get_pos())
        del vs[j]

#Relabel vertices of m so that they match with those of gg
m = Graph(m)
dd0 = dict((k, vs[0])
              for k,vs in dd.items() )
m.relabel(dd0)


#Third: glide to position in m
pos_m = m.get_pos()
pos_g = gg.get_pos()
pics.extend(animate_glide(gg, pos_g, pos_m, frames,
                            vertex_colors = g_colors))
gg.set_pos(pos_m)

#Fourth: delete redundant edges
for e in gg.edges(labels = False):
    if not m.has_edge(e):
        pics.extend(animate_edge_deletion(gg, e, frames,
                            vertex_colors = g_colors))
        gg.delete_edge(*e)
```

```
    #And wait for a moment
    pics.extend([plot(gg, vertex_colors = g_colors)]*frames)

    return animate(pics, xmin = xmin - 0.1, xmax = xmax + 0.1,
                         ymin = ymin - 0.1, ymax = ymax + 0.1)

graph_list = {'CompleteGraph4':graphs.CompleteGraph(4),
              'CompleteGraph5':graphs.CompleteGraph(5),
              'CompleteGraph6':graphs.CompleteGraph(6),
              'BipartiteGraph3,3':graphs.CompleteBipartiteGraph(3,3),
              'BipartiteGraph4,3':graphs.CompleteBipartiteGraph(4,3),
              'PetersenGraph':graphs.PetersenGraph(),
              'CycleGraph4':graphs.CycleGraph(4),
              'CycleGraph5':graphs.CycleGraph(5),
              'BarbellGraph(3,2)':graphs.BarbellGraph(3,2)
              }
@interact
def _(u1 = text_control(value='Does this graph'),
      g  = selector(graph_list.keys(), buttons = True),
      u2 = text_control(value='have a minor isomorphic to this other graph:'),
      m  = selector(graph_list.keys(), buttons = True),
      u3 = text_control(value='''? It is has, show it to me,
      with an animation with the following parameters'''),
      frames = slider(1,15,1,4, label = 'Frames per second'),
      step_time = slider(1/10,2,1/10,1, label = 'Seconds per step')):
    g = graph_list[g]
    m = graph_list[m]
    if g == m:
        html('<h3>Please select two different graphs</h3>')
        return
    try:
        a = animate_minor(g, m, frames = frames)
        a.show(delay=100*step_time/frames)
    except ValueError:
        html('''<h3>The first graph have <strong>NO</strong> minor isomorphic
```

7 Miscellaneous

- Cython: a Python-to-C translator, creates core routines in Sage
- Optimization: CVXOPT, Exact cover with DLX, etc.
- Active community: ∼200 developers, mathematicians and programmers.
- Online servers
- Interfaces to GAP, PARI, Maxima, R, Singular (Python, LaTeX) — all included packages
- Interfaces to Mathematica, Maple, Matlab, Magma, Axiom, Kash, Macaulay2, MuPad, Octave (not included)
- More packages: SymPy, NumPy, Linbox, MPFR, MPFI, NTL, eclib, ATLAS, FLINT, lcalc, PolyBori PyCrypto, matplotlib

8 Philosophy

Mathematica: "Why You Do Not Usually Need to Know about Internals" In the online Mathematica Documentation Center.

> You should realize at the outset that while knowing about the internals of Mathematica may be of intellectual interest, it is usually much less important in practice than you might at first suppose.
>
> ...
>
> Indeed, in almost all practical uses of Mathematica, issues about how Mathematica works inside turn out to be largely irrelevant.
>
> ...
>
> Particularly in more advanced applications of Mathematica, it may sometimes seem worthwhile to try to analyze internal algorithms in order to predict which way of doing a given computation will be the most efficient. And there are indeed occasionally

> major improvements that you will be able to make in specific computations as a result of such analyses.
>
> But most often the analyses will not be worthwhile. For the internals of Mathematica are quite complicated, and even given a basic description of the algorithm used for a particular purpose, it is usually extremely difficult to reach a reliable conclusion about how the detailed implementation of this algorithm will actually behave in particular circumstances.

J. Neubüser (Founder of GAP): "An invitation to computational group theory."
In C. M. Campbell, T. C. Hurley, E. F. Robertson, S. J. Tobin, and J. J. Ward, editors, Groups 93 Galway/St. Andrews, Volume 2, volume 212 of London Mathematical Society Lecture Note Series, pages 457-475. Cambridge University Press, 1995.

> You can read Sylow's Theorem and its proof in Huppert's book in the library without even buying the book and then you can use Sylow's Theorem for the rest of your life free of charge, but... for many computer algebra systems license fees have to be paid regularly for the total time of their use. In order to protect what you pay for, you do not get the source, but only an executable, i.e. a black box. You can press buttons and you get answers in the same way as you get the bright pictures from your television set but you cannot control how they were made in either case.
>
> With this situation two of the most basic rules of conduct in mathematics are violated: In mathematics information is passed on free of charge and everything is laid open for checking. Not applying these rules to computer algebra systems that are made for mathematical research... means moving in a most undesirable direction. Most important: Can we expect somebody to believe a result of a program that he is not allowed to see? Moreover: Do we really want to charge colleagues in Moldava several years of their salary for a computer algebra system?