

# Introduction to the SAGE Graph Class

Robert L. Miller

February 17, 2007

# A GPL Python Library...

- NetworkX

# A GPL Python Library...

- NetworkX
  - A Los Alamos Project:  
<http://networkx.lanl.gov/>

## A GPL Python Library...

- NetworkX
  - A Los Alamos Project:  
<http://networkx.lanl.gov/>
  - Stores graphs as Python dictionaries (a "node-centric dictionary of dictionaries"):

### An example

```
sage: import networkx
sage: K = networkx.complete_graph(3)
sage: K.adj # the adjacency data of the graph
{ 0: {1: None, 2: None},
  1: {0: None, 2: None},
  2: {0: None, 1: None} }
```

# Benefits of NetworkX

## Functionality

centrality, cliques, cluster, component, convert, cores, dag, digraph, distance, drawing, exception, function, generators (atlas, bipartite, classic, degree\_seq, directed, geometric, random\_graphs, small), graph, hybrid, info, io, isomorph, operators, path, queues, release, search, search\_class, spectrum, tests (drawing, generators, test), threshold, tree, utils, xdigraph, xgraph

# Benefits of NetworkX

## Functionality

centrality, cliques, cluster, component, convert, cores, dag, digraph, distance, drawing, exception, function, generators (atlas, bipartite, classic, degree\_seq, directed, geometric, random\_graphs, small), graph, hybrid, info, io, isomorph, operators, path, queues, release, search, search\_class, spectrum, tests (drawing, generators, test), threshold, tree, utils, xdigraph, xgraph

- Fast – Beats out Mathematica (Combinatorica), Maple, and GRAPE (an option of GAP) in every (common to all) function tested

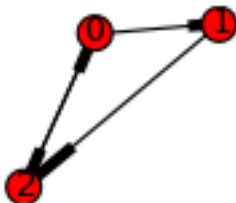
[http://sage.math.washington.edu:9001/graph\\_benchmark](http://sage.math.washington.edu:9001/graph_benchmark)

# Constructing Graphs in SAGE

- Input options:

- NetworkX style dictionary

```
sage: D = DiGraph( { 0: [1, 2],  
...                1: [2],  
...                2: [0] } )  
sage: D.show()
```



# Constructing Graphs in SAGE

- Input options:

- graph6 or sparse6 string

```
sage: s = ":I'AKGsa0s'cI]Gb~"
```

```
sage: G = Graph(s); G
```

Simple graph on 10 vertices (with loops,  
with multiple edges)

# Constructing Graphs in SAGE

- Input options:

- graph6 or sparse6 string

```
sage: s = ":I'AKGsa0s'cI]Gb~"
```

```
sage: G = Graph(s); G
```

Simple graph on 10 vertices (with loops,  
with multiple edges)

- Adjacency matrix

```
sage: M = Matrix ([ [0, 1, 1],
```

```
...                 [1, 0, 1],
```

```
...                 [1, 1, 0] ])
```

```
... # (the order is the number of edges)
```

```
sage: G = Graph(M); G.order()
```

```
3
```

# Constructing Graphs in SAGE

- Input options:

- graph6 or sparse6 string

```
sage: s = ":I'AKGsa0s'cI]Gb~"
sage: G = Graph(s); G
```

Simple graph on 10 vertices (with loops,  
with multiple edges)

- Adjacency matrix

```
sage: M = Matrix ([ [0, 1, 1],
...                 [1, 0, 1],
...                 [1, 1, 0] ])
... # (the order is the number of edges)
sage: G = Graph(M); G.order()
3
```

- Incidence matrix

## Some generators for special graphs

Just type `G = graphs.` and hit `<tab>` to see a list of good graphs to play with. The current list (pending build issues on certain platforms) is

```
BarbellGraph, BullGraph, CircularLadderGraph,  
ClawGraph, CompleteBipartiteGraph, CompleteGraph,  
CubeGraph, CycleGraph, DiamondGraph,  
DodecahedralGraph, EmptyGraph, FlowerSnark,  
FruchtGraph, Grid2dGraph, HeawoodGraph, HouseGraph,  
HouseXGraph, KrackhardtKiteGraph, LadderGraph,  
LollipopGraph, MoebiusKantorGraph, OctahedralGraph,  
PathGraph, PetersenGraph, RandomGNP, RandomGNPFast,  
StarGraph, TetrahedralGraph, ThomsenGraph,  
WheelGraph
```

# Functions

- So far, only the more basic functions have been wrapped. To see them, just do `G = Graph(); G.` and hit `<tab>`. However,

```
sage: P = graphs.PetersenGraph()
sage: N = P.networkx_graph()
sage: import networkx
sage: networkx.any_function(P)
```

gives access to the NetworkX functions, and `G = Graph(N)` will convert NX graphs back to SAGE graphs.

# Functions

- So far, only the more basic functions have been wrapped. To see them, just do `G = Graph(); G.` and hit `<tab>`. However,

```
sage: P = graphs.PetersenGraph()
sage: N = P.networkx_graph()
sage: import networkx
sage: networkx.any_function(P)
```

gives access to the NetworkX functions, and `G = Graph(N)` will convert NX graphs back to SAGE graphs.

- Soon, options will allow a user to specify which program's algorithm (e.g. Magma) to use for each function.

# Plotting

- For a good demo of 2D plotting, see  
[http://sage.math.washington.edu:9001/graph\\_plotting](http://sage.math.washington.edu:9001/graph_plotting)

# Plotting

- For a good demo of 2D plotting, see [http://sage.math.washington.edu:9001/graph\\_plotting](http://sage.math.washington.edu:9001/graph_plotting)
- 3D Plotting
  - Tachyon sprint

# Plotting

- For a good demo of 2D plotting, see [http://sage.math.washington.edu:9001/graph\\_plotting](http://sage.math.washington.edu:9001/graph_plotting)
- 3D Plotting
  - Tachyon sprint
  - Interactive viewing and editing....

## Database Queries

- Jason Grout (at Brigham Young) is working on a SQL database of all graphs on up to nine vertices. SQL is now included with SAGE, and Jason's database is almost finished. When it is, it will be part of the graphs module. For now:

## Database Queries

- Jason Grout (at Brigham Young) is working on a SQL database of all graphs on up to nine vertices. SQL is now included with SAGE, and Jason's database is almost finished. When it is, it will be part of the graphs module. For now:
- Another demo:  
<http://sage.math.washington.edu:22222/>

# Call for Suggestions

Call for suggestions...