NetworkX Tutorial

Jacob Bank (adapted from slides by Evan Rosen)

September 28, 2012

Local Installation

- install manually from
  `http://pypi.python.org/pypi/networkx`
- or use built-in python package manager, easy install
  `$ easy_install networkx`
- or use macports
  `$ sudo port install py27-networkx`
- use pip (replacement for easy_install)
  `$ sudo pip install networkx`
- or use debian package manager
  `$ sudo apt-get install python-networkx`

Using Corn

- networkx is already installed on the corn cluster
- Only works for python version 2.6, 2.7
- However default mapping of command 'python' is to version 2.4
- Just type 'python2.6' instead or make an alias in your shell configuration

Basic Usage

```
>>> import networkx as nx
>>> g = nx.Graph()
>>> g.add_node("spam")
>>> g.add_edge(1,2)
>>> print(g.nodes())
[1, 2, 'spam']
>>> print(g.edges())
[(1, 2)]
```

Graph Types

- Graph : Undirected simple (allows self loops)
- DiGraph : Directed simple (allows self loops)
- MultiGraph : Undirected with parallel edges
- MultiDiGraph : Directed with parallel edges
- can convert to undirected: `g.to_undirected()`
- can convert to directed: `g.to_directed()`

To construct, use standard python syntax:

```
>>> g = nx.Graph()
>>> d = nx.DiGraph()
>>> m = nx.MultiGraph()
>>> h = nx.MultiDiGraph()
```

Adding Nodes

- `add_nodes_from()` takes any iterable collection and any object

```
>>> g = nx.Graph()
>>> g.add_node('a')
>>> g.add_nodes_from( ['b','c','d'])
>>> g.add_nodes_from('xyz')
>>> h = nx.path_graph(5)
>>> g.add_nodes_from(h)
>>> g.nodes()
[0,1,'c','b',4,'d',2,3,5,'x','y','z']
```

Adding Edges

- Adding an edge between nodes that don't exist will automatically add those nodes
- `add_nodes_from()` takes any iterable collection and any type (anything that has a `__iter__()` method)

```
>>> g = nx.Graph( [('a','b'),('b','c'),('c'
    ,'a')] )
>>> g.add_edge('a', 'd')
>>> g.add_edges_from([('d', 'c'), ('d', 'b')
    ])
```

Node Attributes

- Can add node attributes as optional arguments along with most add methods

```
>>> g = nx.Graph()
>>> g.add_node(1,name='Obrian')
>>> g.add_nodes_from([2],name='Quintana'])
>>> g[1]['name']
'Obrian'
```

Edge Attributes

- Can add edge attributes as optional arguments along with most add methods

```
>>> g.add_edge(1, 2, w=4.7 )
>>> g.add_edges_from([(3,4),(4,5)], w =3.0)
>>> g.add_edges_from([(1,2,{'val':2.0})])
# adds third value in tuple as 'weight' attr
>>> g.add_weighted_edges_from([(6,7,3.0)])
>>> g.get_edge_data(3,4)
{'w' : 3.0}
>>> g.add_edge(5,6)
>>> g[5][6]
{}
```

HW0 - Loading the Wikipedia Graph

- We want to load in the Wikipedia graph as a directed graph.

```
>>> file = 'wiki.txt'
>>> wiki = nx.read_adjlist(file, delimiter
    ='\t', create_using=nx.DiGraph())
```

Importing Other Graph Formats

- GML
- Pickle
- GraphML
- YAML
- Pajek
- GEXF
- LEDA
- SparseGraph6
- GIS Shapefile

Simple Graph Generators

- located in `networkx.generators.classic` module
- Complete Graph

```
nx.complete_graph(5)
```

- Chain

```
nx.path_graph(5)
```

- Bipartite

```
nx.complete_bipartite_graph(n1, n2)
```

- Arbitrary Dimensional Lattice (nodes are tuples of ints)

```
nx.grid_graph([10,10,10,10]) # 4D, 100^4
    nodes
```

Random Graph Generators

- located in module `networkx.generators.random_graphs`
- Preferential Attachment

```
nx.barabasi_albert_graph(n, m)
```

- $G_{n,p}$

```
nx.gnp_random_graph(n,p)
```

```
nx.gnm_random_graph(n, m)
```

-

```
nx.watts_strogatz_graph(n, k, p}
```

HW0 - Simple Properties

- Number of nodes :

```
>>> len(wiki)
```

- Number of Self-loops

```
>>> wiki.number_of_selfloops()
```

- Number of Directed Edges

```
>>> wiki.size()
```

- Number of Undirected Edges

```
>>> wiki.to_undirected().size()
```

Degrees

```
 >>> g.degree(0)
1
>>> g.degree([0,1])
{0: 1, 1: 2}
>>> g.degree()
{1: 1, 2: 2, 3: 2, 4: 1}
>>> g.degree().values() # useful for degree
    dist
[1, 2, 2, 1]
```

HW0 - Simple Properties Continued

- Number of Reciprocated Edges :

```
>>> wiki.to_undirected(True).size()
```

- Number of Nodes with OutDegree 0

```
>>> reduce(lambda c, n: c + 1 if wiki.
    out_degree(n) < 1 else c, wiki.nodes()
    , 0)
```

- Number of Nodes with InDegree < 10

```
>>> reduce(lambda c, n: c + 1 if wiki.
    in_degree(n) < 10 else c, wiki.nodes()
    , 0)
```

Neighbors

- Quickly find all of the neighbors of a node.

```
>>> g = nx.Graph()
>>> g.add_edge(1,2)
>>> g.add_edge(2,3)
>>> g.neighbors(2)
[1, 3]
```

Algorithms Package (`networkx.algorithms`)

- bipartite
- block
- boundary
- centrality (package)
- clique
- cluster
- components (package)
- core
- cycles
- dag
- distance_measures
- flow (package)
- isolates
- isomorphism (package)
- link_analysis (package)
- matching
- mixing
- mst
- operators
- shortest_paths (package)
- smetric

Use the Python Help Viewer

```
>>> import networkx as nx
>>> help(nx.algorithms)
```

- pops up an instance of 'less' (the pager utility)

A Few Useful Functions

- As subgraphs

```
nx.connected_component_subgraphs(G)
```

- Operations on Graph

```
nx.union(G,H), intersection(G,H),
    complement(G)
```

- k-cores

```
nx.find_cores(G)
```

A Few More

- shortest path

```
nx.shortest_path(G,s,t)
```

- clustering

```
nx.average_clustering(G)
```

- diameter

```
nx.diameter(G)
```

Matplotlib

- A python package which emulates matlab functionality
  - Well documented at `http://matplotlib.sourceforge.net/contents.html`
- Interfaces nicely with NetworkX
- Depends on Numpy which provides multidimensional array support:
  - `http://numpy.scipy.org/`
- We only really need it for plotting

Setting up Matplotlib

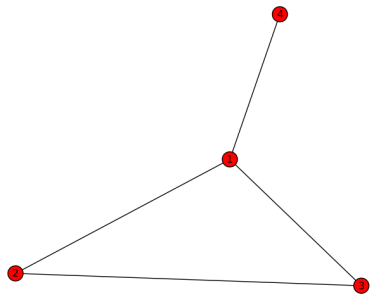
- Need to specify a **backend**, which is the program which is responsible for either displaying or writing the plots to file
- For more info, see: `http://matplotlib.sourceforge.net/faq/installing_faq.html#what-is-a-backend`
- On corn, you simply add the following magic incantation to the top of your python scripts:

```
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
```

Basic Graph Drawing

```
def draw_graph():
    G = nx.Graph()
    G.add_edges_from([(1,2), (2,3), (1,3),
        (1,4)])
    nx.draw(G)
    plt.savefig("simple_graph.png")
```

- consult package `nx.drawing` for more options



Data Plotting - Degree Distribution

First, we find the degree distribution as follows.

```
def plot_degree_distribution():
  degs = {}
  for n in wiki.nodes():
    deg = wiki.degree(n)
    if deg not in degs:
      degs[deg] = 0
    degs[deg] += 1
  items = sorted(degs.items())
```

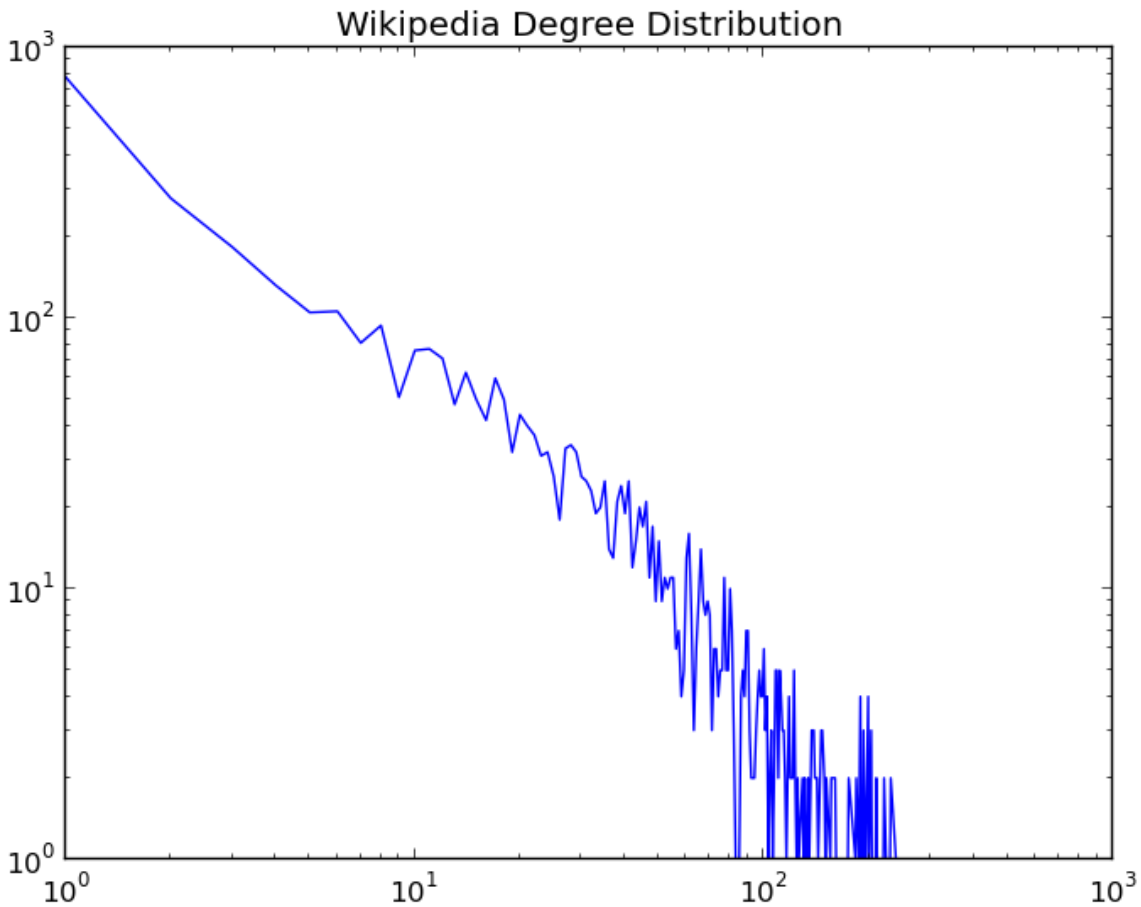
Data Plotting - Degree Distribution continued

Then we plot it.

```
items = sorted(degs.items())
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot([k for (k,v) in items], [v for (k,
    v) in items])
ax.set_xscale('log')
ax.set_yscale('log')
plt.title("Wikipedia Degree Distribution")
fig.savefig("degree_distribution.png")
```

Data Plotting - Degree Distribution continued

And voila!



Resources

- NetworkX Docs
  `http://networkx.lanl.gov/tutorial/index.html`
- NetworkX Tutorial
  `http://networkx.lanl.gov/contents.html`
- Matplotlib Docs
  `http://matplotlib.sourceforge.net/contents.html`
- Matplotlib Tutorial
  `http://matplotlib.sourceforge.net/users/pyplot_tutorial.html`
- Numpy Docs
  `http://numpy.scipy.org/`
- MacPorts
  `http://macports.org`