

# NetworkX

Aric Hagberg

June 4, 2013



Creative Commons Attribution-Share Alike 3.0

1. Introduction and history
2. Getting started
3. Working with data
4. Writing algorithms
5. Live demo
6. The future
7. Questions

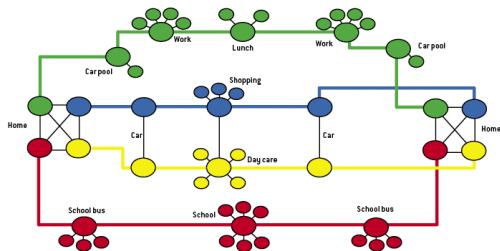
# Introduction

# Example: social networks and epidemics

Understand epidemic outbreak of diseases through modeling

Build social networks from detailed census data

Run dynamic models for smallpox, SARS, flu, etc.

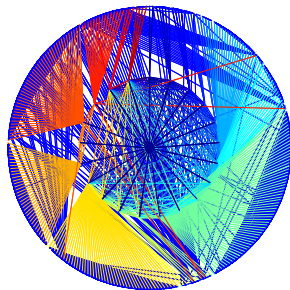


Building a social network

Goal: find a good intervention strategy

If Smallpox Strikes Portland...

by: Chris L. Barrett, Stephen G. Eubank, James P. Smith Scientific American (March 2005)



Social network of one person

## Goals: Why we started project

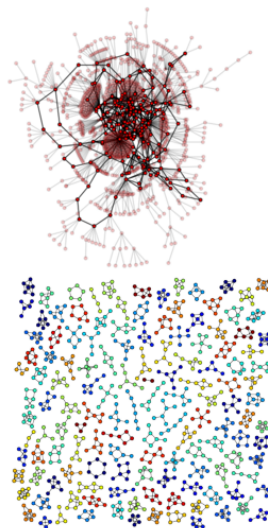
### We needed:

- ▶ Tool to study the structure and dynamics of social, biological, and infrastructure networks
  - ▶ Ease-of-use and rapid development in a collaborative, multidisciplinary environment
  - ▶ Open-source tool base that can easily grow in a multidisciplinary environment with non-expert users and developers
  - ▶ An easy interface to existing code bases written in C, C++, and FORTRAN
  - ▶ To painlessly slurp in large nonstandard data sets
- 
- ▶ No existing API or graph implementation that was suitable
  - ▶ Inspired by Guido van Rossum's 1998 Python graph representation essay
  - ▶ First public release in April 2005

# Features: NetworkX in one slide

Python language package for exploration and analysis of networks and network algorithms

- ▶ Data structures for representing many types of networks, or graphs, (simple graphs, directed graphs, and graphs with parallel edges and self loops)
- ▶ Nodes can be any (hashable) Python object
- ▶ Edges can contain arbitrary data
- ▶ Many network science algorithms (centrality, paths, graph generators)
- ▶ Flexibility ideal for representing networks found in many different fields
- ▶ Many unit and functional tests
- ▶ Online up-to-date documentation
- ▶ Open source software (BSD license), Github developer site
- ▶ Works with Python 3



## NetworkX defines no custom node objects or edge objects

- ▶ “node-centric” view of network
- ▶ Nodes: whatever you put in (hashable) with optional node data
- ▶ Edges: tuples with optional edge data
- ▶ Edge, node data can be anything

## NetworkX is all Python

(Other projects use custom compiled code and Python: Boost Graph, igraph, Graphviz)

- ▶ Focus on computational network modeling not software tool development
- ▶ Move fast to design new algorithms or models

Start Python

Import NetworkX using “nx” as a short name

```
>>> import networkx as nx
```

The basic *Graph* class is used to hold the network information. Nodes can be added as follows:

```
>>> G=nx.Graph()
>>> G.add_node(1) # integer
>>> G.add_node('a') # string
>>> print G.nodes()
['a', 1]
```



## Feature: nodes can be “anything”

Nodes can be any hashable object such as strings, numbers, files, functions, and more

```
>>> import math
>>> G.add_node(math.cos) # cosine function
>>> fh=open('tmp.txt','w')
>>> G.add_node(fh) # file handle
>>> print G.nodes()
[<built-in function cos>,
<open file 'tmp.txt', mode 'w' at 0x30dc38>]
```

## Feature: edges are just pairs of nodes

Edges, or links, between nodes are represented as tuples of nodes. They can be added simply

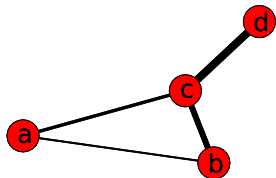
```
>>> G.add_edge(1, 'a')
>>> G.add_edge('b', math.cos)
>>> print G.edges()
[('b', <built-in function cos>), ('a', 1)]
```

If the nodes do not already exist they are automatically added to the graph.

## Feature: Edge can hold arbitrary data

Any Python object is allowed as edge data  
(e.g. number, string, image, file, ip address)  
Edge data assigned and stored in a Python  
dictionary (default empty).

Use Dijkstra's algorithm to find the shortest path:



```
>>> G=nx.Graph()  
>>> G.add_edge('a','b',weight=0.3)  
>>> G.add_edge('b','c',weight=0.5)  
>>> G.add_edge('a','c',weight=2.0)  
>>> G.add_edge('c','d',weight=1.0)  
>>> print nx.shortest_path(G,'a','d')  
['a', 'c', 'd']  
>>> print nx.shortest_path(G,'a','d',weighted=True)  
['a', 'b', 'c', 'd']
```

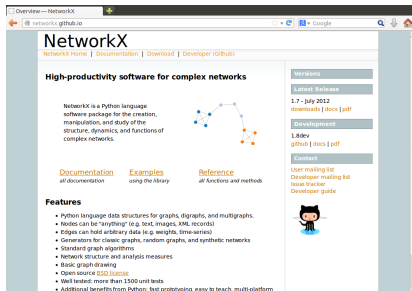
NetworkX has many tests that can be run by users

```
>>> import networkx
>>> networkx.test(verbosity=2)
...
Doctest: networkx.utils ... ok
Conversion from digraph to array to digraph. ... ok
Conversion from digraph to matrix to digraph. ... ok
Conversion from graph to array to graph. ... ok
Conversion from graph to matrix to graph. ... ok
Conversion from weighted digraph to array to weighted digraph. ... ok
...
Conversion from non-square array. ... ok
Conversion from digraph to sparse matrix to digraph. ... ok
Conversion from graph to sparse matrix to graph. ... ok
Conversion from weighted digraph to sparse matrix to weighted digraph. ... ok
Conversion from weighted graph to sparse matrix to weighted graph. ... ok
Conversion from graph to sparse matrix to graph with nodelist. ... ok
Conversion from non-square sparse array. ... ok
Doctest: networkx ... ok

-----
Ran 855 tests in 4.334s

OK
```

# Feature: Online, up-to-date documentation



Overview — NetworkX

NetworkX Home | Documentation | Download | Developer (GitHub)

## NetworkX

High-productivity software for complex networks

NetworkX is a Python language software package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.

[Documentation](#)  
all documentation

[Examples](#)  
using the library

[Reference](#)  
all functions and methods

**Features**

- Python language data structures for graphs, digraphs, and multigraphs.
- Nodes can be "anything" (e.g. text, images, XML records)
- Edges can hold arbitrary data (e.g. weights, time-series)
- Generators for classic graphs, random graphs, and synthetic networks
- Standard graph algorithms
- Network structure and analysis measures
- Basic graph drawing
- Open source [BSD license](#)
- Well tested: more than 1500 unit tests
- Additional benefits from Python: fast prototyping, easy to teach, multi-platform

**Versions**

**Latest Release**


1.7 - July 2012  
[downloads](#) | [docs](#) | [pdf](#)

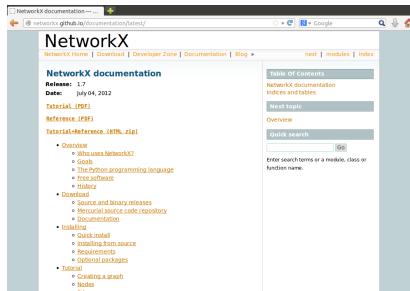
**Development**

1.8dev  
[github](#) | [docs](#) | [pdf](#)

**Contact**

User mailing list  
Developer mailing list  
Issue tracker  
Developer guide





NetworkX documentation —

NetworkX Home | Download | Developer Zone | Documentation | Blog

[next](#) | [modules](#) | [index](#)

## NetworkX

### NetworkX documentation

**Release:** 1.7  
**Date:** July 04, 2012

[Tutorial \(.PDF\)](#)  
[Reference \(.PDF\)](#)  
[Tutorial-Reference \(.HTML.zip\)](#)

- **Overview**
  - Who uses NetworkX?
  - Goals
  - The Python programming language
  - Free software
  - History
- **Download**
  - Source and binary releases
  - Mercurial source code repository
  - Documentation
- **Installing**
  - Quick install
  - Installing from source
  - Requirements
  - Optional packages
- **Tutorial**
  - Creating a graph
  - Nodes
  - Edges

**Table Of Contents**

NetworkX documentation  
Indices and tables

**Next topic**

Overview

**Quick search**

Enter search terms or a module, class or function name.

Python is easy to write and read

### Breadth First Search

```
from collections import deque

def breadth_first_search(g, source):
    queue = deque([(None, source)])
    enqueued = set([source])
    while queue:
        parent, n = queue.popleft()
        yield parent, n
        new = set(g[n]) - enqueued
        enqueued |= new
        queue.extend([(n, child) for child in new])
```

Credit: Matteo Dell'Amico

## Directed Scale-Free Graphs

Béla Bollobás\*

Christian Borgs†

Jennifer Chayes‡

Oliver Riordan§

### 2 The model

We consider a directed graph which grows by adding single edges at discrete time steps. At each such step a vertex may or may not also be added. For simplicity we allow multiple edges and loops. More precisely, let  $\alpha$ ,  $\beta$ ,  $\gamma$ ,  $\delta_{in}$  and  $\delta_{out}$  be non-negative real numbers, with  $\alpha + \beta + \gamma = 1$ . Let  $G_0$  be any fixed initial directed graph, for example a single vertex without edges, and let  $t_0$  be the number of edges of  $G_0$ . (Depending on the parameters, we may have to assume  $t_0 \geq 1$  for the first few steps of our process to make sense.) We set  $G(t_0) = G_0$ , so at time  $t$  the graph  $G(t)$  has exactly  $t$  edges, and a random number  $n(t)$  of vertices. In what follows, to choose a vertex  $v$  of  $G(t)$  according to  $d_{out} + \delta_{out}$  means to choose  $v$  so that  $\Pr(v = v_i)$  is proportional to  $d_{out}(v_i) + \delta_{out}$ , i.e., so that  $\Pr(v = v_i) = (d_{out}(v_i) + \delta_{out}) / (t + \delta_{out}n(t))$ . To choose  $v$  according to  $d_{in} + \delta_{in}$  means to choose  $v$  so that  $\Pr(v = v_i) = (d_{in}(v_i) + \delta_{in}) / (t + \delta_{in}n(t))$ . Here  $d_{out}(v_i)$  and  $d_{in}(v_i)$  are the out-degree and in-degree of  $v_i$ , measured in the graph  $G(t)$ .

For  $t \geq t_0$  we form  $G(t+1)$  from  $G(t)$  according the the following rules:

(A) With probability  $\alpha$ , add a new vertex  $v$  together with an edge from  $v$  to an existing vertex  $w$ , where  $w$  is chosen according to  $d_{in} + \delta_{in}$ .

(B) With probability  $\beta$ , add an edge from an existing vertex  $v$  to an existing vertex  $w$ , where  $v$  and  $w$  are chosen independently,  $v$  according to  $d_{out} + \delta_{out}$ , and  $w$  according to  $d_{in} + \delta_{in}$ .

(C) With probability  $\gamma$ , add a new vertex  $w$  and an edge from an existing vertex  $v$  to  $w$ , where  $v$  is chosen according to  $d_{out} + \delta_{out}$ .

## Feature: Compact code - building new generators

```
1 import bisect
2 import random
3 from networkx import MultiDiGraph
4
5 def scale_free_graph(n, alpha=0.41,beta=0.54,delta_in=0.2,delta_out=0):
6     def _choose_node(G,distribution ,delta ):
7         cumsum = 0.0
8         psum = float(sum(distribution.values()))+float(delta)*len(distribution)
9         r = random.random()
10        for i in range(0, len(distribution)):
11            cumsum += (distribution[i]+delta)/psum
12            if r < cumsum:
13                break
14        return i
15
16    G = MultiDiGraph()
17    G.add_edges_from([(0,1),(1,2),(2,0)])
18    gamma = 1 - alpha - beta
19
20    while len(G)<n:
21        r = random.random()
22        if r < alpha:
23            v = len(G)
24            w = _choose_node(G, G.in_degree(), delta_in)
25        elif r < alpha+beta:
26            v = _choose_node(G, G.out_degree(), delta_out)
27            w = _choose_node(G, G.in_degree(), delta_in)
28        else:
29            v = _choose_node(G, G.out_degree(), delta_out)
30            w = len(G)
31        G.add_edge(v,w)
32    return G
```



## Feature: leveraging libraries

Use well-tested numerical and statistical libraries

E.g. convert Graphs to and from NumPy (and SciPy sparse) matrices

Example: Find eigenvalue spectrum of the graph Laplacian

```
1 >>> L=nx.laplacian(G)
2 >>> print L # a NumPy matrix
3 [[ 1. -1.  0.  0.  0.  0.]
4  [-1.  2. -1.  0.  0.  0.]
5  [ 0. -1.  2. -1.  0.  0.]
6  [ 0.  0. -1.  2. -1.  0.]
7  [ 0.  0.  0. -1.  2. -1.]
8  [ 0.  0.  0.  0. -1.  1.]]
9 >>> import numpy.linalg
10 >>> print numpy.linalg.eigvals(L)
11 [ 3.7321e+00  3.0000e+00  2.0000e+00
12  1.0000e+00 -4.0235e-17  2.6795e-01]
```

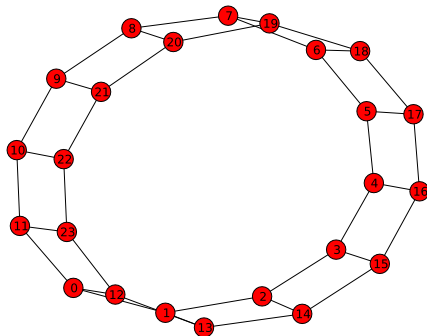


## Feature: drawing

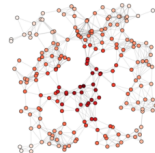
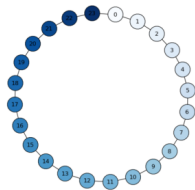
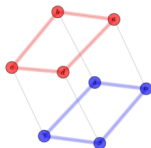
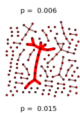
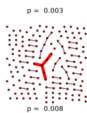
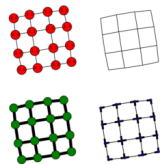
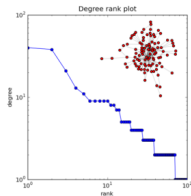
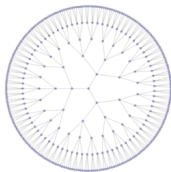
Built-in interface to Matplotlib plotting package

Node positioning algorithms based on force-directed, spectral, and geometric methods

```
>>> G = nx.circular_ladder_graph(12)
>>> nx.draw(G) # Matplotlib under the hood
```



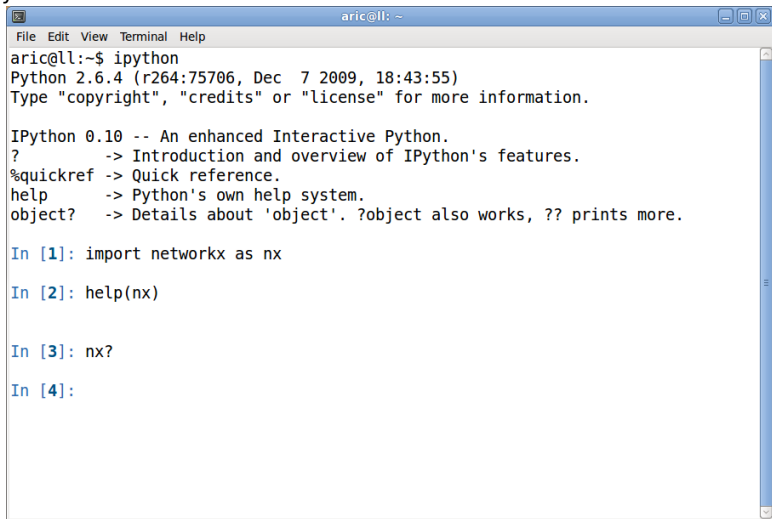
# Drawing with Matplotlib



# Getting Started

- ▶ Running Python and loading NetworkX
- ▶ Creating a Graph, adding nodes and edges
- ▶ Finding what is in NetworkX
- ▶ Interacting with NetworkX graphs
- ▶ Graph generators and operators
- ▶ Basic analysis of graphs

## IPython Command line

A screenshot of a terminal window titled 'aric@ll: ~'. The window has a menu bar with 'File', 'Edit', 'View', 'Terminal', and 'Help'. The terminal output shows the user running 'ipython' at the prompt 'aric@ll:~\$'. This starts Python 2.6.4 and then IPython 0.10. The IPython help text is displayed, listing options like '?', '%quickref', 'help', and 'object?'. The user then enters four IPython commands: 'In [1]: import networkx as nx', 'In [2]: help(nx)', 'In [3]: nx?', and 'In [4]:'.

```
aric@ll:~$ ipython
Python 2.6.4 (r264:75706, Dec 7 2009, 18:43:55)
Type "copyright", "credits" or "license" for more information.

IPython 0.10 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
%quickref  -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object'. ?object also works, ?? prints more.

In [1]: import networkx as nx

In [2]: help(nx)

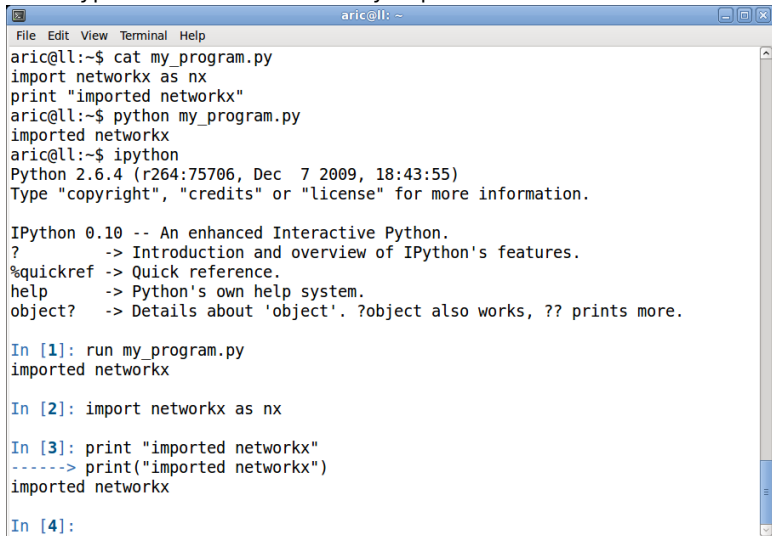
In [3]: nx?

In [4]:
```

No GUI <http://www.cryptonomicon.com/beginning.html>

# Command line vs executing file

You can type commands interactively or put them in a file and run them.



The screenshot shows a terminal window titled 'aric@ll: ~'. The menu bar includes 'File', 'Edit', 'View', 'Terminal', and 'Help'. The terminal content is as follows:

```
aric@ll:~$ cat my_program.py
import networkx as nx
print "imported networkx"
aric@ll:~$ python my_program.py
imported networkx
aric@ll:~$ ipython
Python 2.6.4 (r264:75706, Dec 7 2009, 18:43:55)
Type "copyright", "credits" or "license" for more information.

IPython 0.10 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
%quickref  -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object'. ?object also works, ?? prints more.

In [1]: run my_program.py
imported networkx

In [2]: import networkx as nx

In [3]: print "imported networkx"
-----> print("imported networkx")
imported networkx

In [4]:
```

The basic *Graph* object is used to hold the network information.  
Create an empty graph with no nodes and no edges:

```
1 >>> import networkx as nx
2
3 >>> G = nx.Graph()
```

The graph *G* can be grown in several ways.  
NetworkX includes many graph generator functions and facilities to read and write graphs in many formats.

```
1 # One node at a time
2 >>> G.add_node(1) # "method" of G
3
4 # A list of nodes
5 >>> G.add_nodes_from([2,3])
6
7 # A container of nodes
8 >>> H = nx.path_graph(10)
9 >>> G.add_nodes_from(H) # G now contains the nodes of H
10
11 # In contrast, you could use the graph H as a node in G.
12 >>> G.add_node(H) # G now contains Graph H as a node
```

Nodes can be any hashable object such as strings, numbers, files, functions, and more.



G can also be grown by adding edges.

```
1 # Single edge
2 >>> G.add_edge(1,2)
3 >>> e = (2,3)
4 >>> G.add_edge(*e) # unpack edge tuple*
5
6 # List of edges
7
8 >>> G.add_edges_from([(1,2),(1,3)])
9
10 # Container of edges
11 >>> G.add_edges_from(H.edges())
```

If the nodes do not already exist they are automatically added to the graph.  
You can demolish the graph similarly with

G.remove\_node, G.remove\_nodes\_from,  
G.remove\_edge, G.remove\_edges\_from.

- ▶ How do I find out the names of the methods like `add_edge`?
- ▶ How do I see what is in my graph?

# What's in NetworkX?

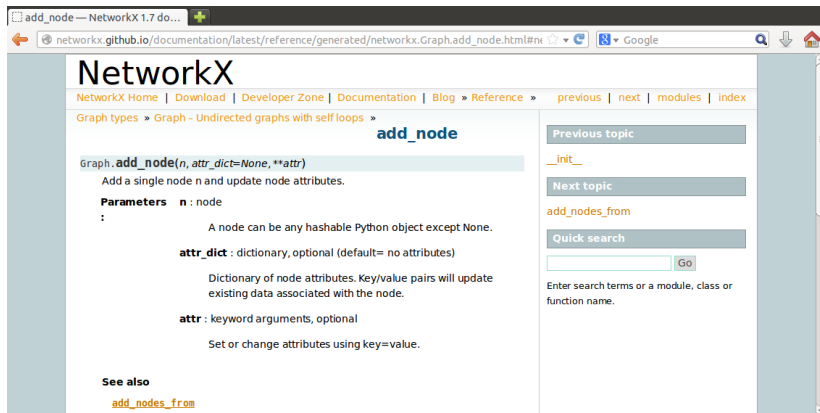
Graph - Undirected graphs wit... +

networkx.github.io/documentation/latest/reference/classes.graph.html#adding-and-removing-nodes

## Adding and removing nodes and edges

<code>Graph.__init__([data])</code>	Initialize a graph with edges, name, graph attributes.
<code>Graph.add_node(n[, attr_dict])</code>	Add a single node n and update node attributes.
<code>Graph.add_nodes_from(nodes, **attr)</code>	Add multiple nodes.
<code>Graph.remove_node(n)</code>	Remove node n.
<code>Graph.remove_nodes_from(nodes)</code>	Remove multiple nodes.
<code>Graph.add_edge(u, v[, attr_dict])</code>	Add an edge between u and v.
<code>Graph.add_edges_from(ebunch[, attr_dict])</code>	Add all the edges in ebunch.
<code>Graph.add_weighted_edges_from(ebunch[, weight])</code>	Add all the edges in ebunch as weighted edges with specified weights.
<code>Graph.remove_edge(u, v)</code>	Remove the edge between u and v.
<code>Graph.remove_edges_from(ebunch)</code>	Remove all edges specified in ebunch.
<code>Graph.add_star(nodes, **attr)</code>	Add a star.
<code>Graph.add_path(nodes, **attr)</code>	Add a path.
<code>Graph.add_cycle(nodes, **attr)</code>	Add a cycle.
<code>Graph.clear()</code>	Remove all nodes and edges from the graph.

# What's in NetworkX?



The screenshot shows a web browser window displaying the NetworkX documentation for the `add_node` function. The browser's address bar shows the URL `networkx.github.io/documentation/latest/reference/generated/networkx.Graph.add_node.html#n`. The page has a navigation bar with links to [NetworkX Home](#), [Download](#), [Developer Zone](#), [Documentation](#), [Blog](#), and [Reference](#). The [Reference](#) link is active, and the breadcrumb trail shows [Graph types](#) » [Graph - Undirected graphs with self loops](#) » **`add_node`**.

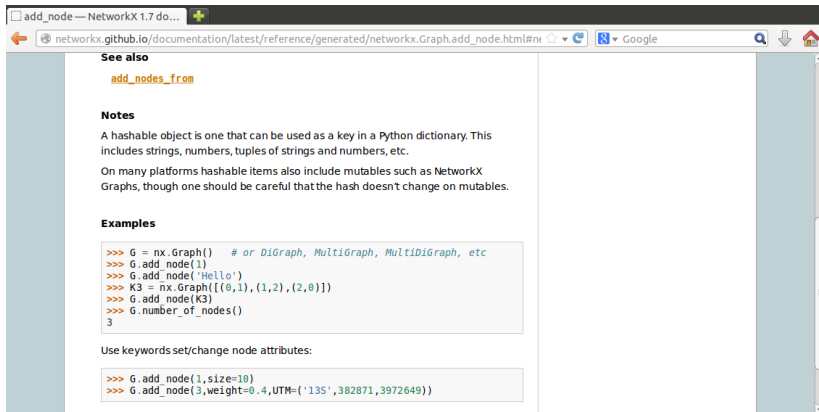
The main content area describes the `Graph.add_node(n, attr_dict=None, **attr)` function. It states: "Add a single node n and update node attributes." The parameters are listed as follows:

- Parameters**
  - n** : node
    - A node can be any hashable Python object except None.
  - attr\_dict** : dictionary, optional (default= no attributes)
    - Dictionary of node attributes. Key/value pairs will update existing data associated with the node.
  - attr** : keyword arguments, optional
    - Set or change attributes using key=value.

Under the heading "See also", there is a link to [add\\_nodes\\_from](#).

On the right side of the page, there is a sidebar with navigation links: "Previous topic" (linking to `_init_`), "Next topic" (linking to `add_nodes_from`), and a "Quick search" section with a text input field and a "Go" button. Below the search bar, it says: "Enter search terms or a module, class or function name."

# What's in Networkx?



The screenshot shows a web browser window with the address bar displaying `networkx.github.io/documentation/latest/reference/generated/networkx.Graph.add_node.html#networkx.Graph.add_node`. The page content includes:

- See also**
  - [add\\_nodes\\_from](#)
- Notes**

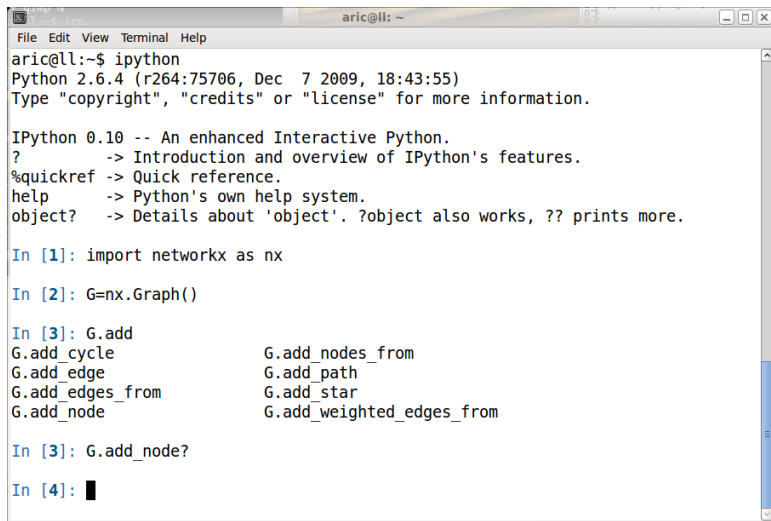
A hashable object is one that can be used as a key in a Python dictionary. This includes strings, numbers, tuples of strings and numbers, etc.

On many platforms hashable items also include mutables such as NetworkX Graphs, though one should be careful that the hash doesn't change on mutables.
- Examples**

```
>>> G = nx.Graph() # or DiGraph, MultiGraph, MultiDiGraph, etc
>>> G.add_node(1)
>>> G.add_node('Hello')
>>> K3 = nx.Graph([(0,1),(1,2),(2,0)])
>>> G.add_node(K3)
>>> G.number_of_nodes()
3
```
- Use keywords set/change node attributes:

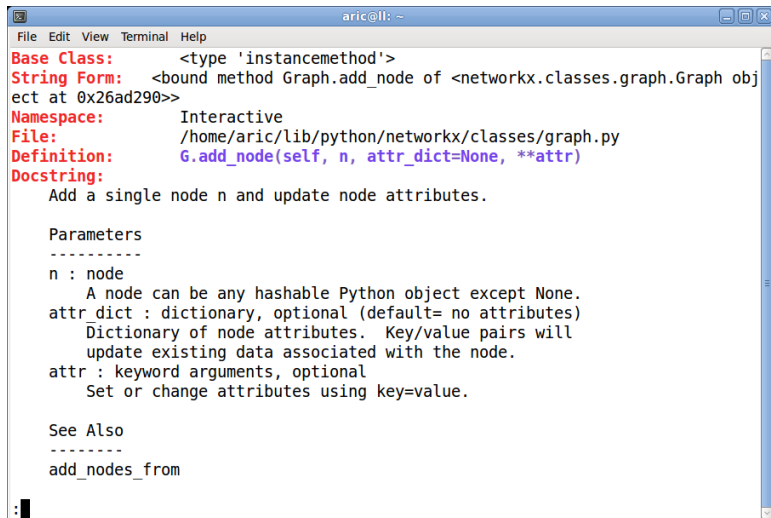
```
>>> G.add_node(1,size=10)
>>> G.add_node(3,weight=0.4,UTM=('13S',382871,3972649))
```

# What's in Networkx?



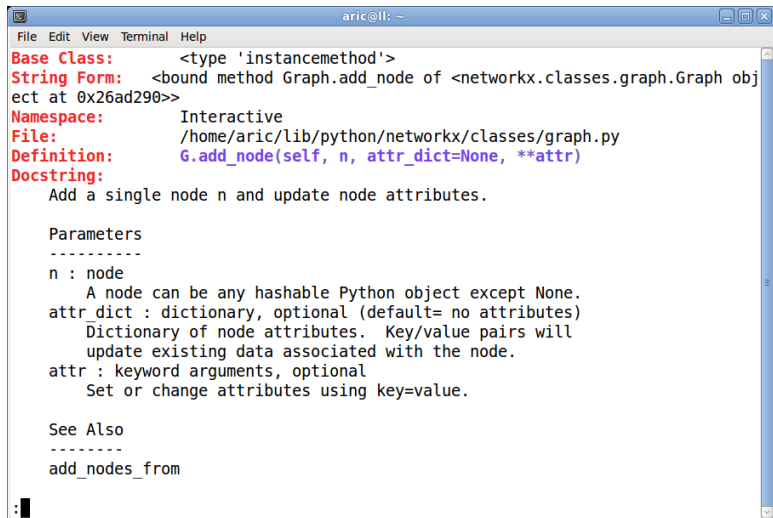
```
aric@ll: ~  
File Edit View Terminal Help  
aric@ll:~$ ipython  
Python 2.6.4 (r264:75706, Dec 7 2009, 18:43:55)  
Type "copyright", "credits" or "license" for more information.  
  
IPython 0.10 -- An enhanced Interactive Python.  
? -> Introduction and overview of IPython's features.  
%quickref -> Quick reference.  
help -> Python's own help system.  
object? -> Details about 'object'. ?object also works, ?? prints more.  
  
In [1]: import networkx as nx  
  
In [2]: G=nx.Graph()  
  
In [3]: G.add  
G.add_cycle          G.add_nodes_from  
G.add_edge           G.add_path  
G.add_edges_from     G.add_star  
G.add_node           G.add_weighted_edges_from  
  
In [3]: G.add_node?  
  
In [4]: █
```

# What's in Networkx?



```
aric@ll: ~  
File Edit View Terminal Help  
Base Class:      <type 'instancemethod'>  
String Form:    <bound method Graph.add_node of <networkx.classes.graph.Graph object at 0x26ad290>>  
Namespace:      Interactive  
File:           /home/aric/lib/python/networkx/classes/graph.py  
Definition:     G.add_node(self, n, attr_dict=None, **attr)  
Docstring:  
    Add a single node n and update node attributes.  
  
    Parameters  
    -----  
    n : node  
        A node can be any hashable Python object except None.  
    attr_dict : dictionary, optional (default= no attributes)  
        Dictionary of node attributes. Key/value pairs will  
        update existing data associated with the node.  
    attr : keyword arguments, optional  
        Set or change attributes using key=value.  
  
    See Also  
    -----  
    add_nodes_from  
:  
█
```

# What's in Networkx?



```
aric@ll: ~  
File Edit View Terminal Help  
Base Class:          <type 'instancemethod'>  
String Form:       <bound method Graph.add_node of <networkx.classes.graph.Graph object at 0x26ad290>>  
Namespace:         Interactive  
File:              /home/aric/lib/python/networkx/classes/graph.py  
Definition:        G.add_node(self, n, attr_dict=None, **attr)  
Docstring:  
    Add a single node n and update node attributes.  
  
    Parameters  
    -----  
    n : node  
        A node can be any hashable Python object except None.  
    attr_dict : dictionary, optional (default= no attributes)  
        Dictionary of node attributes. Key/value pairs will  
        update existing data associated with the node.  
    attr : keyword arguments, optional  
        Set or change attributes using key=value.  
  
    See Also  
    -----  
    add_nodes_from  
:  
:
```

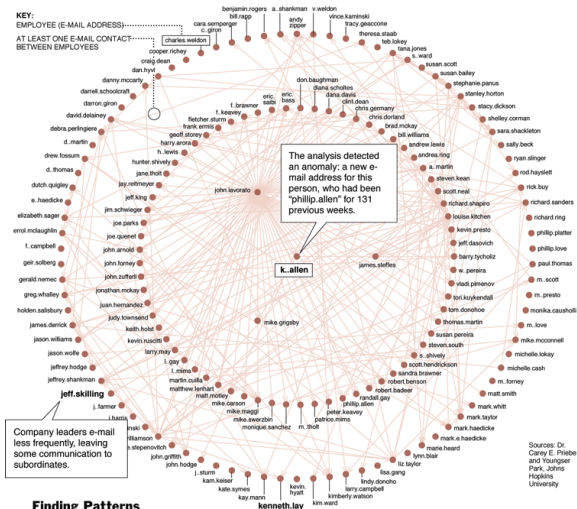
Demo



# Adding attributes to graphs, nodes, and edges

(Almost) any Python object is allowed as graph, node, and edge data.

- ▶ number
- ▶ string
- ▶ image
- ▶ IP address
- ▶ email address



## Finding Patterns In Corporate Chatter

Computer scientists are analyzing about a half million Enron e-mails. Here is a map of a week's e-mail patterns in May 2001, when a new name suddenly appeared. Scientists found that this week's pattern differed greatly from others, suggesting different conversations were taking place that might interest investigators. Next step: word analysis of these messages.

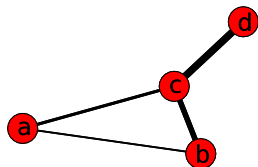
```
1 >>> import networkx as nx
2 # Assign graph attributes when creating a new graph
3
4 >>> G = nx.Graph(day="Friday")
5 >>> G.graph
6 {'day': 'Friday'} # Python dictionary
7
8 # Or you can modify attributes later
9
10 >>> G.graph['day'] = 'Monday'
11 >>> G.graph
12 {'day': 'Monday'}
```

```
1
2 # Add node attributes using add_node(), add_nodes_from() or G.node
3 >>> G.add_node(1, time='5pm')
4 >>> G.node[1]['time']
5 '5pm'
6 >>> G.node[1] # Python dictionary
7 {'time': '5pm'}
8
9 >>> G.add_nodes_from([3], time='2pm') # multiple nodes
10 >>> G.node[1]['room'] = 714 # add new attribute
11
12 >>> G.nodes(data=True)
13 [(1, {'room': 714, 'time': '5pm'}), (3, {'time': '2pm'})]
```

```
1 # Add edge attributes using add_edge(), add_edges_from(),  
2 # subscript notation, or G.edge.  
3 >>> G.add_edge(1, 2, weight=4.0 )  
4 >>> G[1][2]['weight'] = 4.0 # edge already added  
5 >>> G.edge[1][2]['weight'] = 4.0 # edge already added  
6  
7 >>> G[1][2]['weight']  
8 4.0  
9 >>> G[1][2]  
10 {'weight': 4.0}  
11  
12 >>> G.add_edges_from([(3,4),(4,5)], color='red')  
13 >>> G.add_edges_from([(1,2,{'color': 'blue'})], (2,3,{'weight':8})))  
14  
15 >>> G.edges()  
16 [(1, 2), (2, 3), (3, 4), (4, 5)]  
17 >>> G.edges(data=True)  
18 [(1, 2, {'color': 'blue', 'weight': 4.0}), (2, 3, {'weight': 8}), (3,
```

## Weighted graph example

The special attribute 'weight' holds values used by algorithms requiring weighted edges.



Use Dijkstra's algorithm to find the shortest path:

```
1 >>> G=nx.Graph()
2 >>> G.add_edge('a','b',weight=0.3)
3 >>> G.add_edge('b','c',weight=0.5)
4 >>> G.add_edge('a','c',weight=2.0)
5 >>> G.add_edge('c','d',weight=1.0)
6 >>> print nx.shortest_path(G,'a','d')
7 ['a', 'c', 'd']
8 >>> print nx.shortest_path(G,'a','d',weighted=True)
9 ['a', 'b', 'c', 'd']
```

### Applying classic graph operations

`subgraph(G, nbunch)` - induce subgraph of G on nodes in nbunch

`union(G1, G2)` - graph union

`disjoint_union(G1, G2)` - graph union assuming all nodes are different

`cartesian_product(G1, G2)` - return Cartesian product graph

`compose(G1, G2)` - combine graphs identifying nodes common to both

`complement(G)` - graph complement

`create_empty_copy(G)` - return an empty copy of the same graph class

`convert_to_undirected(G)` - return an undirected representation of G

`convert_to_directed(G)` - return a directed representation of G

```
1 # small graphs
2 petersen = nx.petersen_graph()
3 tutte = nx.tutte_graph()
4 maze = nx.sedgewick_maze_graph()
5 tet = nx.tetrahedral_graph()
6
7 # classic graphs
8 K_5 = nx.complete_graph(5)
9 K_3_5 = nx.complete_bipartite_graph(3,5)
10 barbell = nx.barbell_graph(10,10)
11 lollipop = nx.lollipop_graph(10,20)
12
13 # random graphs
14 er = nx.erdos_renyi_graph(100,0.15)
15 ws = nx.watts_strogatz_graph(30,3,0.1)
16 ba = nx.barabasi_albert_graph(100,5)
17 red = nx.random_lobster(100,0.9,0.9)
```

```
1 >>> G=nx.Graph()
2 >>> G.add_edges_from([(1,2),(1,3)])
3 >>> G.add_node("spam")
4
5 # Structure of G can be analyzed using various
6 # graph-theoretic functions
7 >>> nx.connected_components(G)
8 [[1, 2, 3], ['spam']]
9
10 # Functions that return node properties return
11 # dictionaries keyed by node label.
12 >>> nx.degree(G)
13 {1: 2, 2: 1, 3: 1, 'spam': 0}
14
15 >>> sorted(nx.degree(G).values())
16 [0, 1, 1, 2]
17
18 >>> nx.clustering(G)
19 {1: 0.0, 2: 0.0, 3: 0.0, 'spam': 0.0}
```



# Working with data

Read a graph stored in a file using common graph formats.

- edge lists

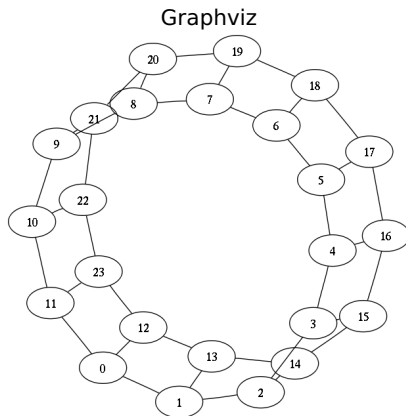
- adjacency lists

  - GML

  - GraphML

  - Pajek

  - LEDA



Output to: dot, GML, LEDA, edge list, adjacency list, YAML, sparsegraph6, GraphML

# Writing Algorithms

- ▶ Examples of some simple algorithms
- ▶ Writing a new algorithm

Python is easy to write and read

### Breadth First Search

```
1 from collections import deque
2
3 def breadth_first_search(g, source):
4     queue = deque([(None, source)])
5     enqueued = set([source])
6     while queue:
7         parent, n = queue.popleft()
8         yield parent, n
9         new = set(g[n]) - enqueued
10        enqueued |= new
11        queue.extend([(n, child) for child in new])
```

Credit: Matteo Dell'Amico

Python is easy to write and read

### Erdős-Rényi Random graph

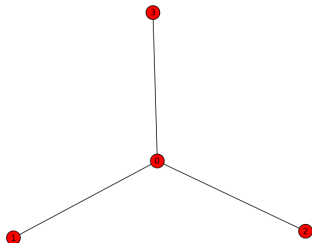
```
1 from itertools import combinations
2 from random import random
3 def gnp_random_graph(n, p):
4     G=empty_graph(n) # NetworkX
5     edges=combinations(range(n),2)
6     for e in edges:
7         if random() < p:
8             G.add_edge(*e) # NetworkX
9     return G
```

### Erdős-Rényi Random graph

```
1 def fast_gnp_random_graph(n, p):
2     G = empty_graph(n)
3     G.name="fast_gnp_random_graph(%s,%s)"%(n,p)
4
5     if not seed is None:
6         random.seed(seed)
7
8     if p <= 0 or p >= 1:
9         return nx.gnp_random_graph(n,p)
10
11     v = 1 # Nodes in graph are from 0,n-1
12     w = -1
13     lp = math.log(1.0 - p)
14     while v < n:
15         lr = math.log(1.0 - random.random())
16         w = w + 1 + int(lr/lp)
17         while w >= v and v < n:
18             w = w - v
19             v = v + 1
20         if v < n:
21             G.add_edge(v, w)
22     return G
```

For a graph with  $n$  nodes

$$C_D(v) = \frac{\deg(v)}{n - 1}$$



```
1 >>> G=nx.star_graph(3)
2 >>> print G.edges()
3 [(0, 1), (0, 2), (0, 3)]
4 >>> print G.degree(0)
5 3
6 >>> print len(G) # # of nodes
7 4
8 >>> print G.degree(0)/3
9 0
10 >>> print G.degree(0)/3.0
11 1
12 >>> for v in G:
13 ...     print v, G.degree(v)/3.0
14 0 1.0
15 1 0.3333333333333333
16 2 0.3333333333333333
17 3 0.3333333333333333
```



## Degree centrality 1

```
1 import networkx as nx
2
3 def degree centrality(G):
4
5     n = len(G) - 1.0 # forces floating point for n
6     for v in G:
7         print v, G.degree(v)/n
8
9     return
10
11 G = nx.star_graph(3)
12 degree centrality(G)
13 # 0 1.0
14 # 1 0.333333333333
15 # 2 0.333333333333
16 # 3 0.333333333333
```

## Degree centrality 2

```
1 import networkx as nx
2
3 def degree_centrality(G):
4
5     centrality = {} # empty dictionary
6     n = len(G) - 1.
7     for v in G:
8         centrality[v] = G.degree(v)/n
9     return centrality
10
11 G = nx.star_graph(3)
12 dc = degree_centrality(G)
13 for v in dc:
14     print v,dc[v]
15 # 0 1.0
16 # 1 0.3333333333333333
17 # 2 0.3333333333333333
18 # 3 0.3333333333333333
19 print dc
20 # {0: 1.0, 1: 0.33333, 2: 0.33333, 3: 0.33333}
```

```
1 def degree centrality(G):
2     centrality = {} # empty dictionary
3     n = len(G)-1.0 # forces floating point for n
4     for v in G:
5         centrality[v] = G.degree(v)/n
6     return centrality
7
8 if __name__=='__main__':
9     import networkx as nx
10    G = nx.star_graph(3)
11    for v,c in degree_centrality(G).items():
12        print v,c
13 # 0 1.0
14 # 1 0.333333333333
15 # 2 0.333333333333
16 # 3 0.333333333333
```

```
1 def degree centrality(G):
2     """Compute degree centrality for nodes.
3
4     The degree centrality for a node is the fraction of all other
5     nodes it is connected to.
6
7     >>> import networkx as nx
8     >>> G = nx.star_graph(3)
9     >>> print degree centrality(G)[0]
10    1.0
11    """
12    centrality = {} # empty dictionary
13    n = len(G)-1.0 # forces floating point for n
14    for v in G:
15        centrality[v] = G.degree(v)/n
16    return centrality
```

# Degree centrality in NetworkX

```
1 def degree centrality(G):
2     """Compute the degree centrality for nodes.
3
4     The degree centrality for a node  $v$  is the fraction of nodes it
5     is connected to.
6
7     Parameters
8     -----
9     G : graph
10         A Networkx graph
11
12     Returns
13     -----
14     nodes : dictionary
15         Dictionary of nodes with degree centrality as the value.
16
17     See Also
18     -----
19     betweenness centrality, load centrality, eigenvector centrality
20
21     Notes
22     -----
23     The degree centrality values are normalized by dividing by the maximum
24     possible degree in a simple graph  $n-1$  where  $n$  is the number of nodes in  $G$ .
25
26     For multigraphs or graphs with self loops the maximum degree might
27     be higher than  $n-1$  and values of degree centrality greater than 1
28     are possible.
29     """
30     s = 1.0/(len(G)-1.0)
31     return dict((n,d*s) for n,d in G.degree_iter())
```

# Demo

# Future

approx 100K downloads of networkx-1.7

The screenshot shows the PyPI package page for networkx 1.7. The browser address bar shows the URL https://pypi.python.org/pypi/networkx/. The page features the Python 3 logo and a search bar. A left sidebar contains a 'PACKAGE INDEX' menu with links like 'Browse packages', 'Package submission', and 'List packages'. The main content area displays the package name 'networkx 1.7' and a description: 'Python package for creating and manipulating graphs and networks'. A 'Downloads' button is visible. Below this is a table listing available download files. A 'Not Logged In' box on the right offers links for 'Login', 'Register', 'Lost Login?', and 'Use OpenID'. At the bottom, there is a search bar with the word 'download' and navigation links like 'Previous', 'Next', 'Highlight all', and 'Match case'.

networkx 1.7: Python Package...

https://pypi.python.org/pypi/networkx/

python 3

» Package Index > networkx > 1.7

**PACKAGE INDEX** »

- Browse packages
- Package submission
- List trove classifiers
- List packages
- RSS (latest 40 updates)
- RSS (newest 40 packages)
- Python 3 Packages
- PyPI Tutorial
- PyPI Security
- PyPI Support
- PyPI Bug Reports
- PyPI Discussion
- PyPI Developer Info

ABOUT »

NEWS »

DOCUMENTATION »

DOWNLOAD »

COMMUNITY »

FOUNDATION »

CORE DEVELOPMENT »

LINKS »

## networkx 1.7

*Python package for creating and manipulating graphs and networks*

Downloads

NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.

Not Logged In

- Login
- Register
- Lost Login?
- Use OpenID

File	Type	Py Version	Uploaded on	Size
<a href="#">networkx-1.7-py2.6.egg (md5)</a>	Python Egg	2.6	2012-07-05	1MB
<a href="#">networkx-1.7-py2.7.egg (md5)</a>	Python Egg	2.7	2012-07-05	1MB
<a href="#">networkx-1.7-py3.2.egg (md5)</a>	Python Egg	3.2	2012-07-05	1MB
<a href="#">networkx-1.7.tar.gz (md5)</a>	Source		2012-07-05	711KB
<a href="#">networkx-1.7.zip (md5)</a>	Source		2012-07-05	1009KB

**Author:** NetworkX Developers  
**Home Page:** <http://networkx.lanl.gov/>  
**Download URL:** <http://networkx.lanl.gov/download/networkx>  
**Keywords:** Networks, Graph Theory, Mathematics, network, graph, discrete mathematics, math  
**License:** BSD  
**Platform:** Linux, Mac OSX, Windows, Unix  
**Categories:** [Development Status :: 4 - Beta](#)

Find:  [Previous](#) [Next](#) [Highlight all](#) ☐ Match case



## networkx-discuss Google group 1130 members

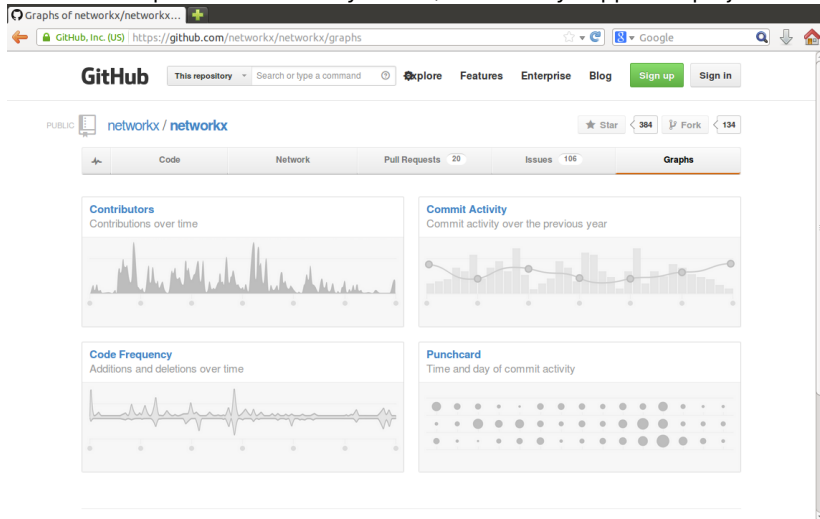
The screenshot shows a web browser window displaying the 'networkx-discuss' Google Group forum. The browser's address bar shows the URL 'https://groups.google.com/forum/#!forum/networkx-discuss'. The Google search bar is visible with the text 'Search for topics' and a 'SIGN IN' button. Below the search bar, the 'Groups' section is active, showing a list of topics. The first topic, 'Nomenclature and Methods', is selected and highlighted. The list of topics includes their titles, authors, post counts, and view counts, along with the time of the last post.

Topic Title	Author	Posts	Views	Last Post Time
Nomenclature and Methods	By ray	1 post	2 views	12:26 PM
write_gexf: trouble coloring nodes/edges	By Jared Hawkins	3 posts	3 views	May 22
Hierarchical Graphs in NetworkX	By Stephan Gerhard	4 posts	91 views	May 17
All shortest paths for weighted graphs?	By Federico Battiston	3 posts	5 views	May 15
How to install pygraphviz on windows	By Federico Vaggi	7 posts	476 views	May 14
Adding Code to The Repository	By Fred Morstatter	3 posts	11 views	May 9
Lowest common ancestor?	By Pau Rullian Ferragut	6 posts	35 views	May 8
How to draw networkx graph with edge labels	By Dushyant	4 posts	971 views	May 7
Divide by zero in layout.py when calling spring_layout.				

<http://networkx.github.io/>

networkx-1.8 release soon

Active development: community driven, community supported project.



We hope you will contribute (after class is fine).

- ▶ More algorithms (contribute)
- ▶ Better interaction with graph drawing tools
- ▶ Inline integration with IPython notebook
- ▶ Community finding algorithms

# Questions?