NetworkX

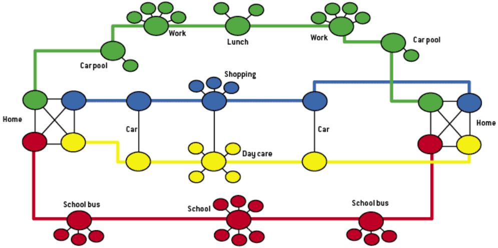
Aric Hagberg

June 4, 2013

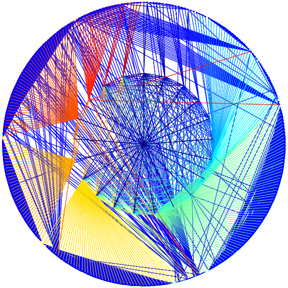
1. Introduction and history
2. Getting started
3. Working with data
4. Writing algorithms
5. Live demo
6. The future
7. Questions

Introduction

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



Social network of one person

Goal: find a good intervention strategy
If Smallpox Strikes Portland...

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

Goals: Why we started project

We needed:

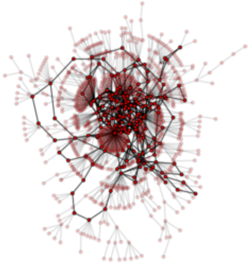
- Tool to study the structure and dynamics of social, biological, and infrastructure networks
- Ease-of-use and rapid development in a collaborative, multidisciplinary environment
- Open-source tool base that can easily grow in a multidisciplinary environment with non-expert users and developers
- An easy interface to existing code bases written in C, C++, and FORTRAN
- To painlessly slurp in large nonstandard data sets

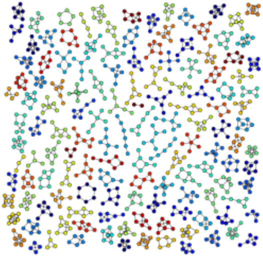
- No existing API or graph implementation that was suitable
- Inspired by Guido van Rossum's 1998 Python graph representation essay
- First public release in April 2005

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





Design decisions

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, graph-tool

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

Feature: Simple use, adding nodes

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

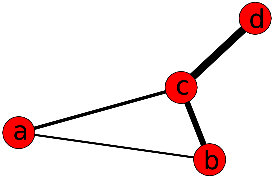
```
>>> G.add_edge(1,'a')
>>> G.add_edge('b',math.cos)
>>> print G.edges()
[('b', <built-in function cos>), ('a', 1)]
```

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:



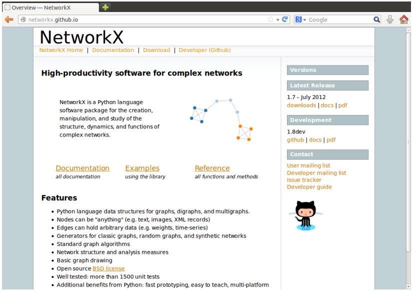

```
>>> G=nx.Graph()
>>> G.add_edge('a','b',weight=0.3)
>>> G.add_edge('b','c',weight=0.5)
>>> G.add_edge('a','c',weight=2.0)
>>> G.add_edge('c','d',weight=1.0)
>>> print nx.shortest_path(G,'a','d')
['a', 'c', 'd']
>>> print nx.shortest_path(G,'a','d',weighted=True)
['a', 'b', 'c', 'd']
```

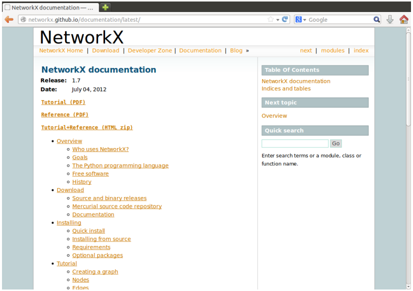
Feature: testing

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 1798 tests in 17.202s

OK
```

Feature: Online, up-to-date documentation





Feature: Python expressivity - a simple algorithm

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

Feature: Compact code - building new generators

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 α, β, γ, δ_{in} and δ_{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 α, 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 β, 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 γ, add a new vertex w and an edge from an existing vertex v to w, where v is chosen according to $d_{out} + \delta_{out}$.

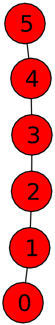
Feature: Compact code - building new generators

```python
import bisect
import random
from networkx import MultiDiGraph

def scale_free_graph(n, alpha=0.41,beta=0.54,delta_in=0.2,delta_out=0):
    def _choose_node(G,distribution,delta):
        cumsum = 0.0
        psum = float(sum(distribution.values()))+float(delta)*len(distribution)
        r = random.random()
        for i in range(0, len(distribution)):
            cumsum += (distribution[i]+delta)/psum
            if r < cumsum:
                break
        return i

    G = MultiDiGraph()
    G.add_edges_from([(0,1),(1,2),(2,0)])
    gamma = 1 - alpha - beta

    while len(G)<n:
        r = random.random()
        if r < alpha:
            v = len(G)
            w = _choose_node(G, G.in_degree(),delta_in)
        elif r < alpha+beta:
            v = _choose_node(G, G.out_degree(),delta_out)
            w = _choose_node(G, G.in_degree(),delta_in)
        else:
            v = _choose_node(G, G.out_degree(),delta_out)
            w = len(G)
        G.add_edge(v,w)
    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

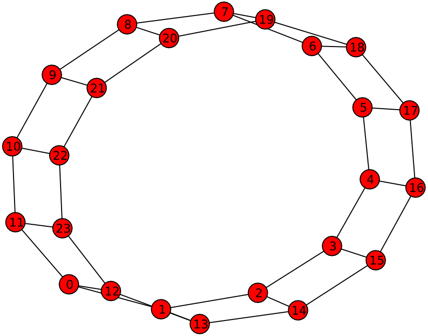
```
>>> L=nx.laplacian(G)
>>> print L  # a NumPy matrix
[[ 1. -1.  0.  0.  0.  0.]
 [-1.  2. -1.  0.  0.  0.]
 [ 0. -1.  2. -1.  0.  0.]
 [ 0.  0. -1.  2. -1.  0.]
 [ 0.  0.  0. -1.  2. -1.]
 [ 0.  0.  0.  0. -1.  1.]]
>>> import numpy.linalg
>>> print numpy.linalg.eigvals(L)
[  3.7321e+00   3.0000e+00   2.0000e+00
   1.0000e+00  -4.0235e-17   2.6795e-01]
```

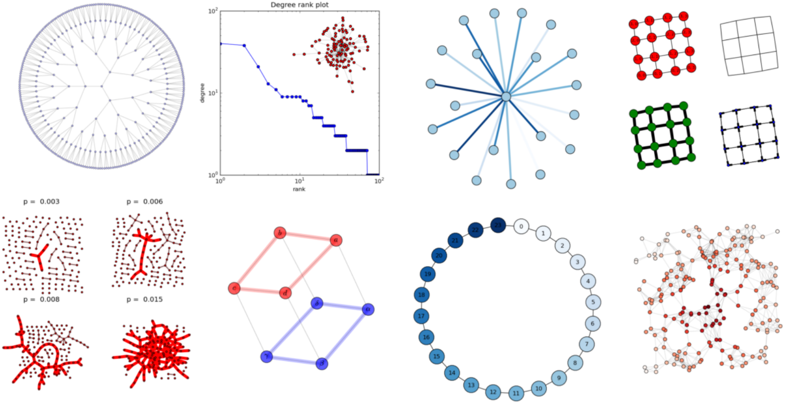


Feature: drawing

- 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
```




Degree rank plot
degree
rank
p = 0.003
p = 0.006
p = 0.008
p = 0.015

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

Running Python and loading NetworkX

IPython Command line

```
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]: 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.

```
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]:
```

Creating a graph

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

```
>>> import networkx as nx

>>> 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.

Adding nodes

```
# One node at a time
>>> G.add_node(1)  # "method" of G

# A list of nodes
>>> G.add_nodes_from([2,3])

# A container of nodes
>>> H = nx.path_graph(10)
>>> G.add_nodes_from(H) # G now contains the nodes of H

# In contrast, you could use the graph H as a node in G.
>>> 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.

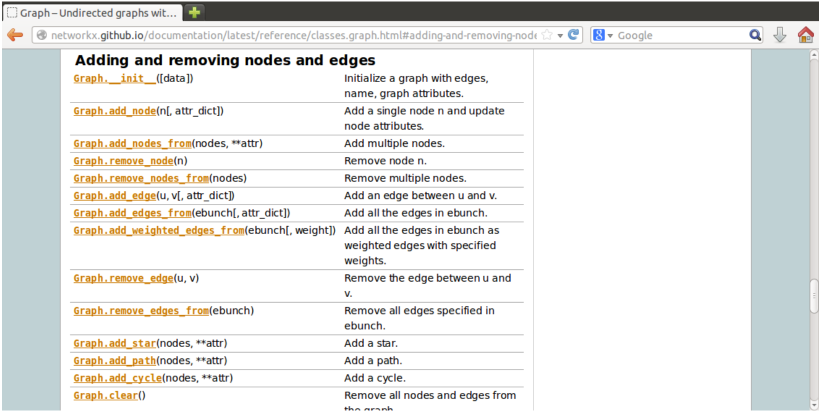
```
# Single edge
>>> G.add_edge(1,2)
>>> e = (2,3)
>>> G.add_edge(*e) # unpack edge tuple*

# List of edges

>>> G.add_edges_from([(1,2),(1,3)])

# Container of edges
>>> 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?

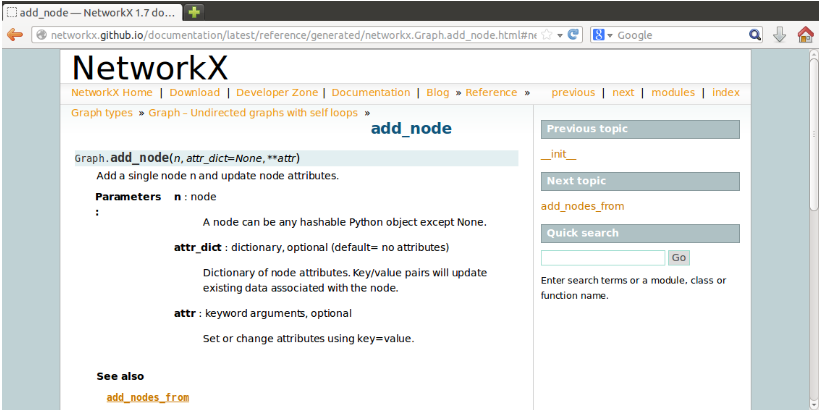
What’s in NetworkX?



Adding and removing nodes and edges

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

What’s in NetworkX?



What’s in Networkx?

add_node — NetworkX 1.7 do...

networkx.github.io/documentation/latest/reference/generated/networkx.Graph.add_node.html#ne

Google

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:~$ 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 obj
ect 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

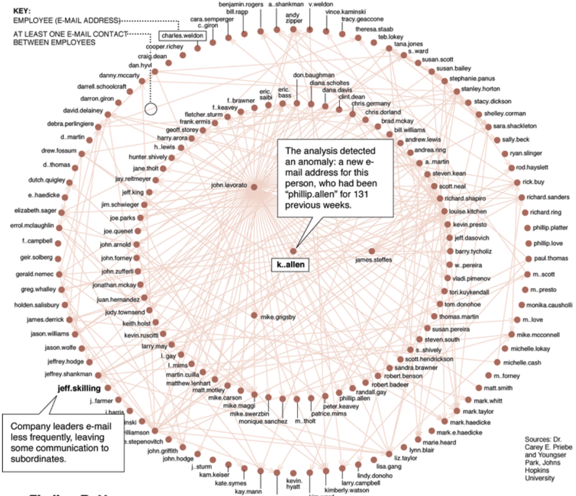
```
Base Class:       <type 'instancemethod'>
String Form:    <bound method Graph.add_node of <networkx.classes.graph.Graph obj
ect at 0x26ad290>>
Namespace:        Interactive
File:             /home/aric/lib/python/networkx/classes/graph.py
Definition:       G.add_node(self, n, attr_dict=None, **attr)
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.

Graph attributes

```
>>> import networkx as nx
# Assign graph attributes when creating a new graph

>>> G = nx.Graph(day="Friday")
>>> G.graph
{'day': 'Friday'} # Python dictionary

# Or you can modify attributes later

>>> G.graph['day']='Monday'
>>> G.graph
{'day': 'Monday'}
```

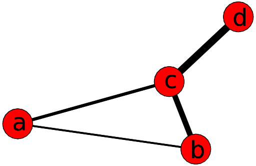
```
# Add node attributes using add_node(), add_nodes_from() or G.node
>>> G.add_node(1, time='5pm')
>>> G.node[1]['time']
'5pm'
>>> G.node[1] # Python dictionray
{'time': '5pm'}

>>> G.add_nodes_from([3], time='2pm') # multiple nodes
>>> G.node[1]['room'] = 714 # add new attribute

>>> G.nodes(data=True)
[(1, {'room': 714, 'time': '5pm'}), (3, {'time': '2pm'})]
```

Edge attributes

```
# Add edge attributes using add_edge(), add_edges_from(),
# subscript notation, or G.edge.
>>> G.add_edge(1, 2, weight=4.0 )
>>> G[1][2]['weight'] = 4.0 # edge already added
>>> G.edge[1][2]['weight'] = 4.0 # edge already added

>>> G[1][2]['weight']
4.0
>>> G[1][2]
{'weight': 4.0}

>>> G.add_edges_from([(3,4),(4,5)], color='red')
>>> G.add_edges_from([(1,2,{'color':'blue'}), (2,3,{'weight':8})])

>>> G.edges()
[(1, 2), (2, 3), (3, 4), (4, 5)]
>>> G.edges(data=True)
[(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:

```
>>> 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']
```

More ways to build graphs: operators and generators

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

Call a graph generator

```
# small graphs
petersen = nx.petersen_graph()
tutte = nx.tutte_graph()
maze = nx.sedgewick_maze_graph()
tet = nx.tetrahedral_graph()

# classic graphs
K_5 = nx.complete_graph(5)
K_3_5 = nx.complete_bipartite_graph(3,5)
barbell = nx.barbell_graph(10,10)
lollipop = nx.lollipop_graph(10,20)

# random graphs
er = nx.erdos_renyi_graph(100,0.15)
ws = nx.watts_strogatz_graph(30,3,0.1)
ba = nx.barabasi_albert_graph(100,5)
red = nx.random_lobster(100,0.9,0.9)
```

Basic analysis of graphs

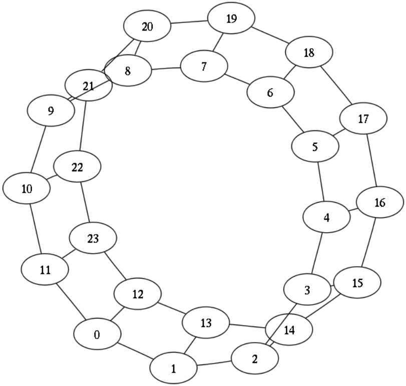
```
>>> G=nx.Graph()
>>> G.add_edges_from([(1,2),(1,3)])
>>> G.add_node("spam")

# Structure of G can be analyzed using various
# graph-theoretic functions
>>> nx.connected_components(G)
[[1, 2, 3], ['spam']]

# Functions that return node properties return
# dictionaries keyed by node label.
>>> nx.degree(G)
{1: 2, 2: 1, 3: 1, 'spam': 0}

>>> sorted(nx.degree(G).values())
[0, 1, 1, 2]

>>> nx.clustering(G)
{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

Graphviz



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

```
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

Python is easy to write and read

Erdős-Rényi Random graph

```
from itertools import combinations
from random import random
def gnp_random_graph(n, p):
    G=empty_graph(n)   # NetworkX
    edges=combinations(range(n),2)
    for e in edges:
        if random() < p:
            G.add_edge(*e) # NetworkX
    return G
```

Feature: Python expressivity - a simple algorithm

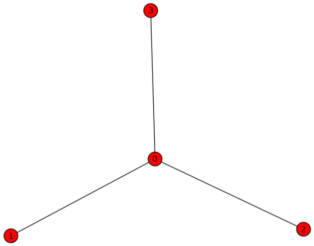
Erdős-Rényi Random graph

```python
def fast_gnp_random_graph(n, p):
    G = empty_graph(n)
    G.name="fast_gnp_random_graph(%s,%s)"%(n,p)

    if not seed is None:
        random.seed(seed)

    if p <= 0 or p >= 1:
        return nx.gnp_random_graph(n,p)

    v = 1  # Nodes in graph are from 0,n-1
    w = -1
    lp = math.log(1.0 - p)
    while v < n:
        lr = math.log(1.0 - random.random())
        w = w + 1 + int(lr/lp)
        while w >= v and v < n:
            w = w - v
            v = v + 1
        if v < n:
            G.add_edge(v, w)
    return G
```

Degree centrality

For a graph with n nodes

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




```
>>> G=nx.star_graph(3)
>>> print G.edges()
[(0, 1), (0, 2), (0, 3)]
>>> print G.degree(0)
3
>>> print len(G) # # of nodes
4
>>> print G.degree(0)/3
0
>>> print G.degree(0)/3.0
1
>>> for v in G:
...     print v, G.degree(v)/3.0
0 1.0
1 0.333333333333
2 0.333333333333
3 0.333333333333
```

```python
import networkx as nx

def degree_centrality(G):

    n = len(G) - 1.0 # forces floating point for n
    for v in G:
        print v,G.degree(v)/n

    return

G = nx.star_graph(3)
degree_centrality(G)
# 0 1.0
# 1 0.333333333333
# 2 0.333333333333
# 3 0.333333333333
```

Degree centrality 2

```python
import networkx as nx

def degree_centrality(G):

    centrality = {} # empty dictionary
    n = len(G) - 1.
    for v in G:
        centrality[v] = G.degree(v)/n
    return centrality

G = nx.star_graph(3)
dc = degree_centrality(G)
for v in dc:
    print v,dc[v]
# 0 1.0
# 1 0.333333333333
# 2 0.333333333333
# 3 0.333333333333
print dc
# {0: 1.0, 1: 0.33333, 2: 0.33333, 3: 0.33333}
```

Degree centrality 3

```
def degree_centrality(G):
    centrality = {} # empty dictionary
    n = len(G)−1.0 # forces floating point for n
    for v in G:
        centrality[v] = G.degree(v)/n
    return centrality

if __name__=='__main__':
    import networkx as nx
    G = nx.star_graph(3)
    for v,c in degree_centrality(G).items():
        print v,c
# 0 1.0
# 1 0.333333333333
# 2 0.333333333333
# 3 0.333333333333
```

```python
def degree_centrality(G):
    """Compute degree centrality for nodes.

    The degree centrality for a node is the fraction of all other
    nodes it is connected to.

    >>> import networkx as nx
    >>> G = nx.star_graph(3)
    >>> print degree_centrality(G)[0]
    1.0
    """
    centrality = {} # empty dictionary
    n = len(G)-1.0 # forces floating point for n
    for v in G:
        centrality[v] = G.degree(v)/n
    return centrality
```

Degree centrality in NetworkX

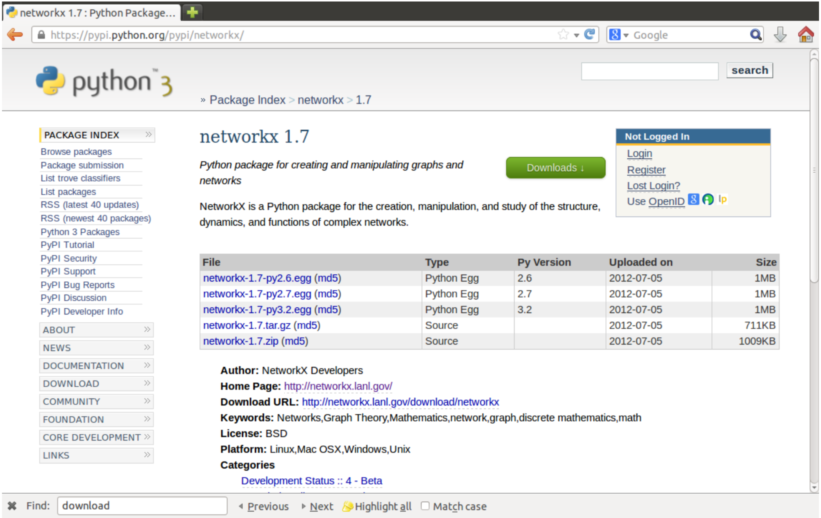
```python
def degree_centrality(G):
    """Compute the degree centrality for nodes.

    The degree centrality for a node v is the fraction of nodes it
    is connected to.

    Parameters
    ----------
    G : graph
      A Networkx graph

    Returns
    -------
    nodes : dictionary
       Dictionary of nodes with degree centrality as the value.

    See Also
    --------
    betweenness_centrality, load_centrality, eigenvector_centrality

    Notes
    -----
    The degree centrality values are normalized by dividing by the maximum
    possible degree in a simple graph n-1 where n is the number of nodes in G.

    For multigraphs or graphs with self loops the maximum degree might
    be higher than n-1 and values of degree centrality greater than 1
    are possible.
    """
    s = 1.0/(len(G)-1.0)
    return dict((n,d*s) for n,d in G.degree_iter())
```

Demo

Future

Download

approx 100K downloads of networkx-1.7



networkx 1.7 : Python Package...

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

python 3

search

» 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?

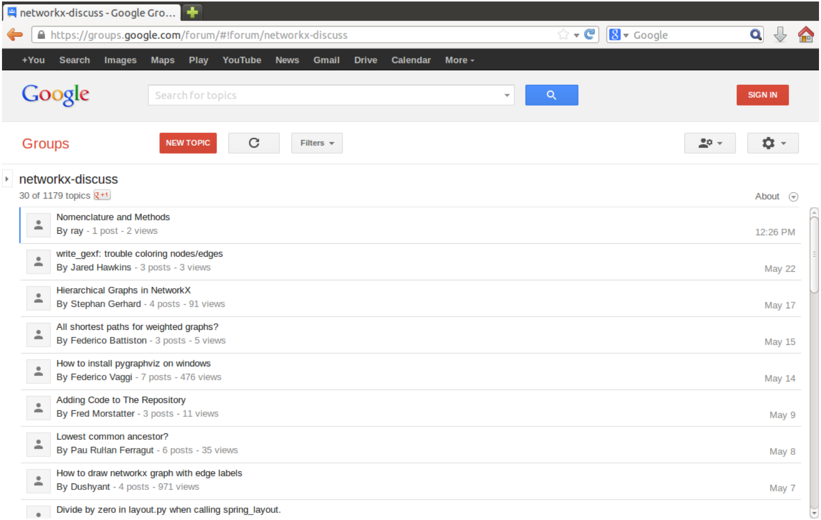
Use OpenID

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

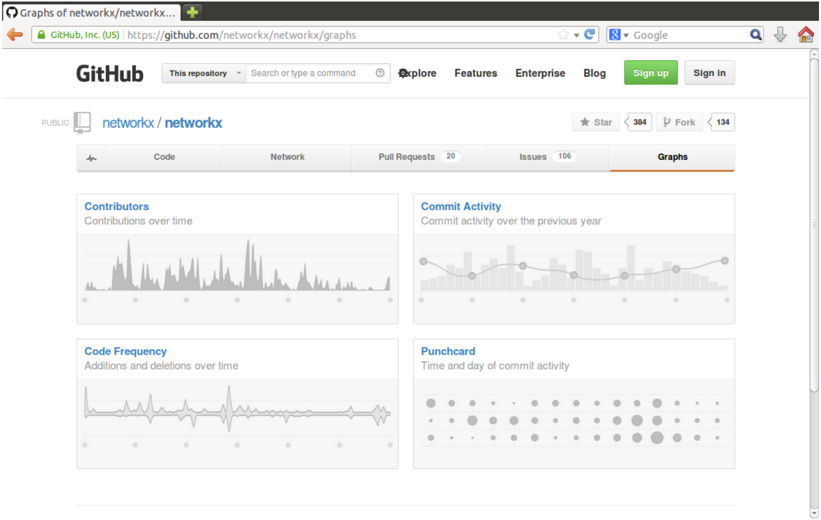
Find: download ◂ Previous ▸ Next Highlight all Match case networkx-discuss Google group 1130 members



`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).

New features

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

Questions?