1 - Introduction to NetworkX

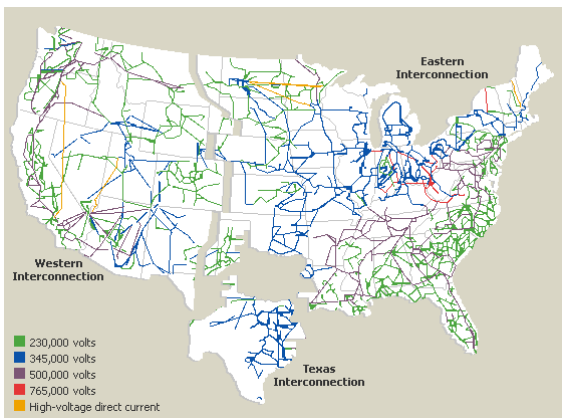
Drew Conway and Aric Hagberg

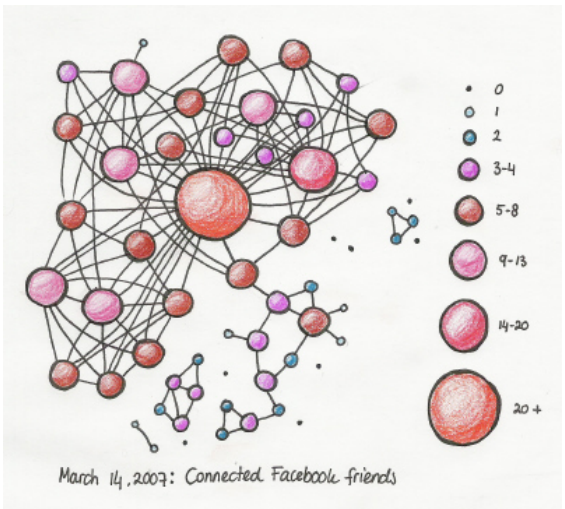
June 29, 2010

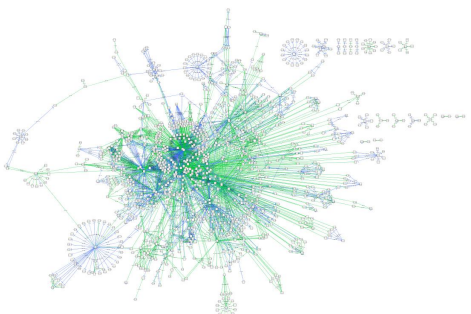
Networks

Vast amounts of data are being generated and collected

- Sociology: WWW, email, social networking
- Technology: Internet, telecommunications, power grid
- Biology: protein interactions, genetic regulatory networks, epidemiology



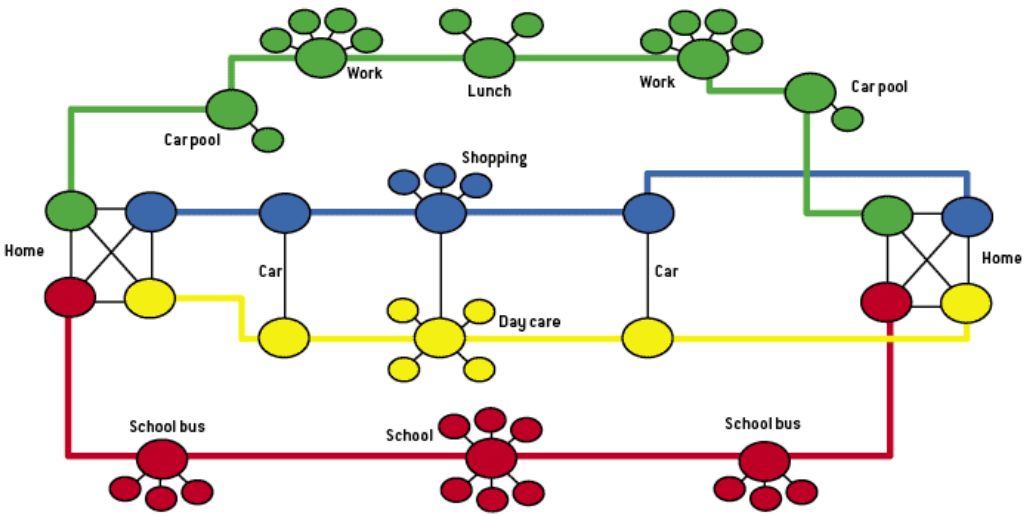




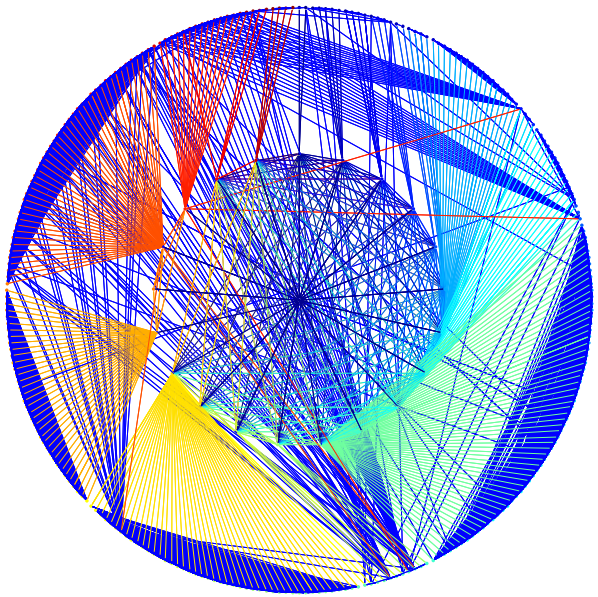
Need theory, analysis, models

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



Social network of one person

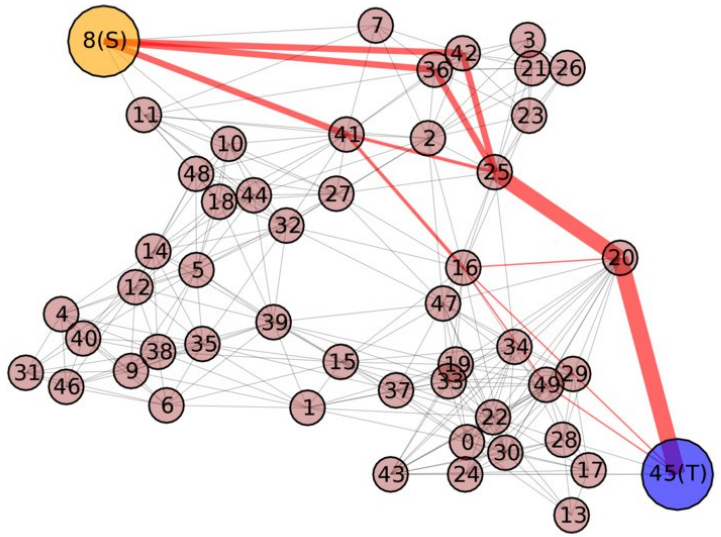
Goal: find a good intervention strategy

NISAC: EpiSimS

Example: interdiction

Problem: smuggling of nuclear material in transportation network

Detector at border crossing



Find best set of roads (edges) to monitor (cut) with limited budget

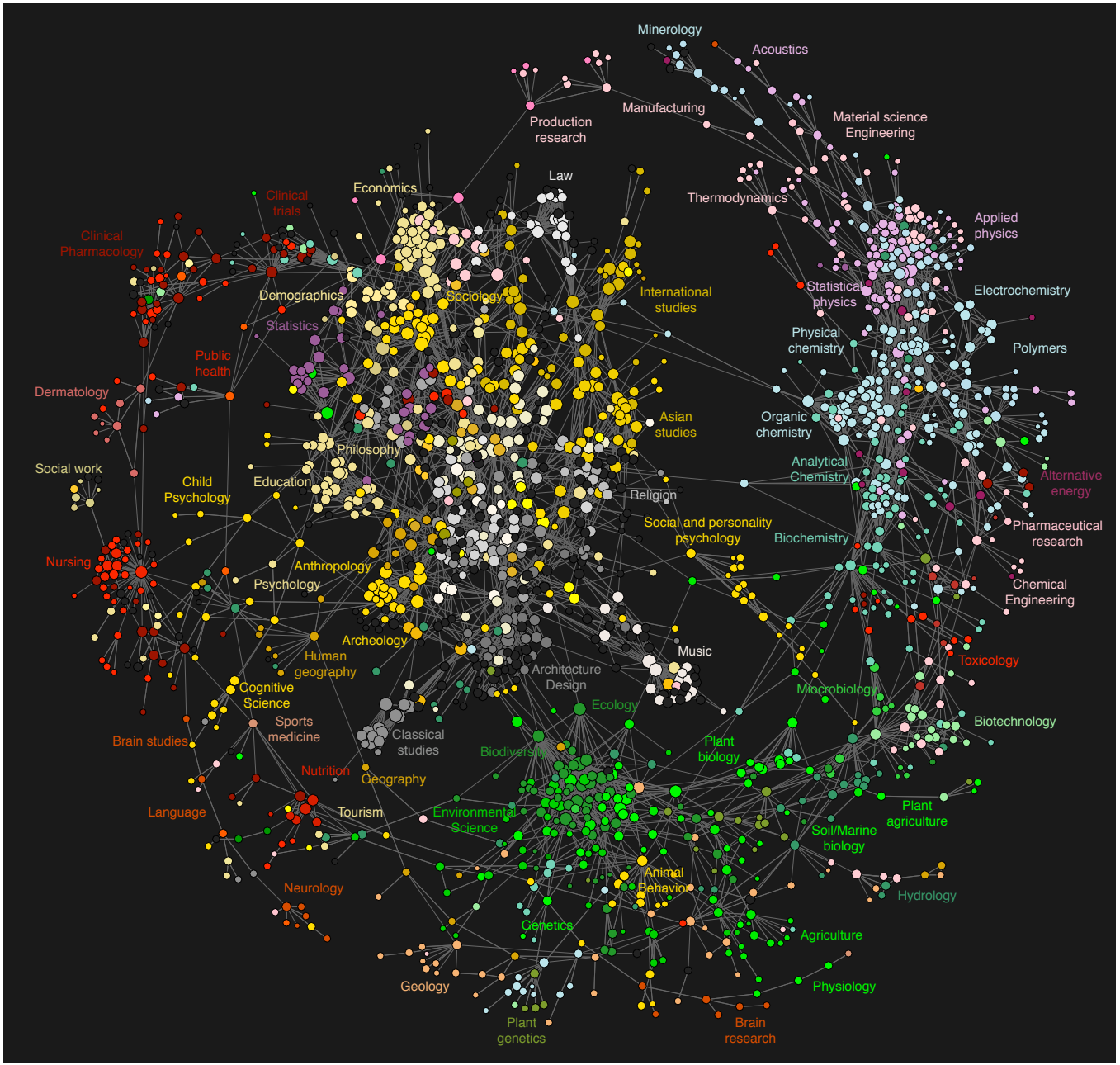
Example: journal usage network

University libraries, journals, and aggregators collect journal usage data through web portals
MESUR project is analyzing about 1 billion usage events
Build network from user click streams

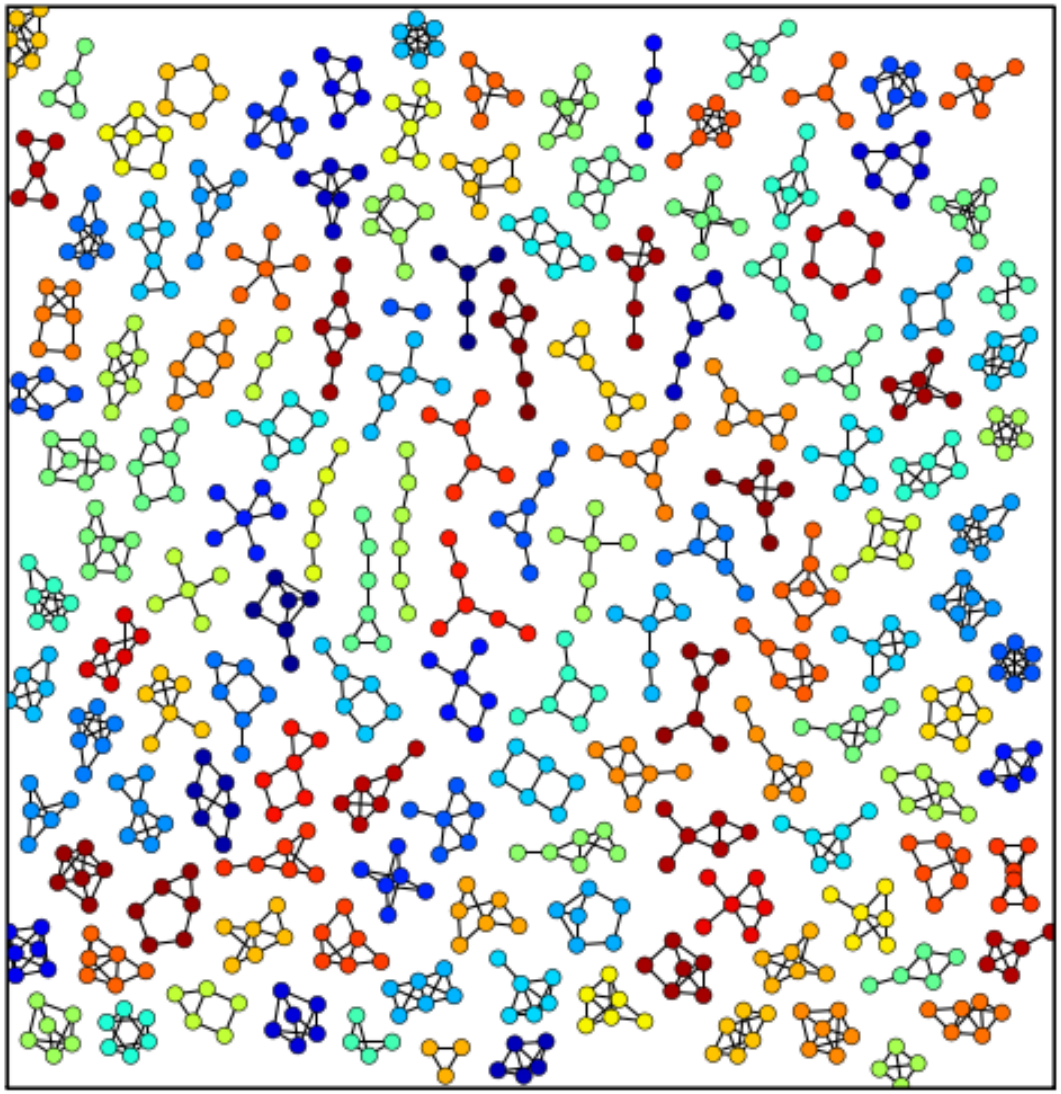
- Do scholars read and cite journals in the same way?
- Can new trends in research (new field, interdisciplinary) be spotted?
- Which journals are most important according to usage?

Johan Bollen, Los Alamos

Example: journal usage network



NetworkX project: goals and features



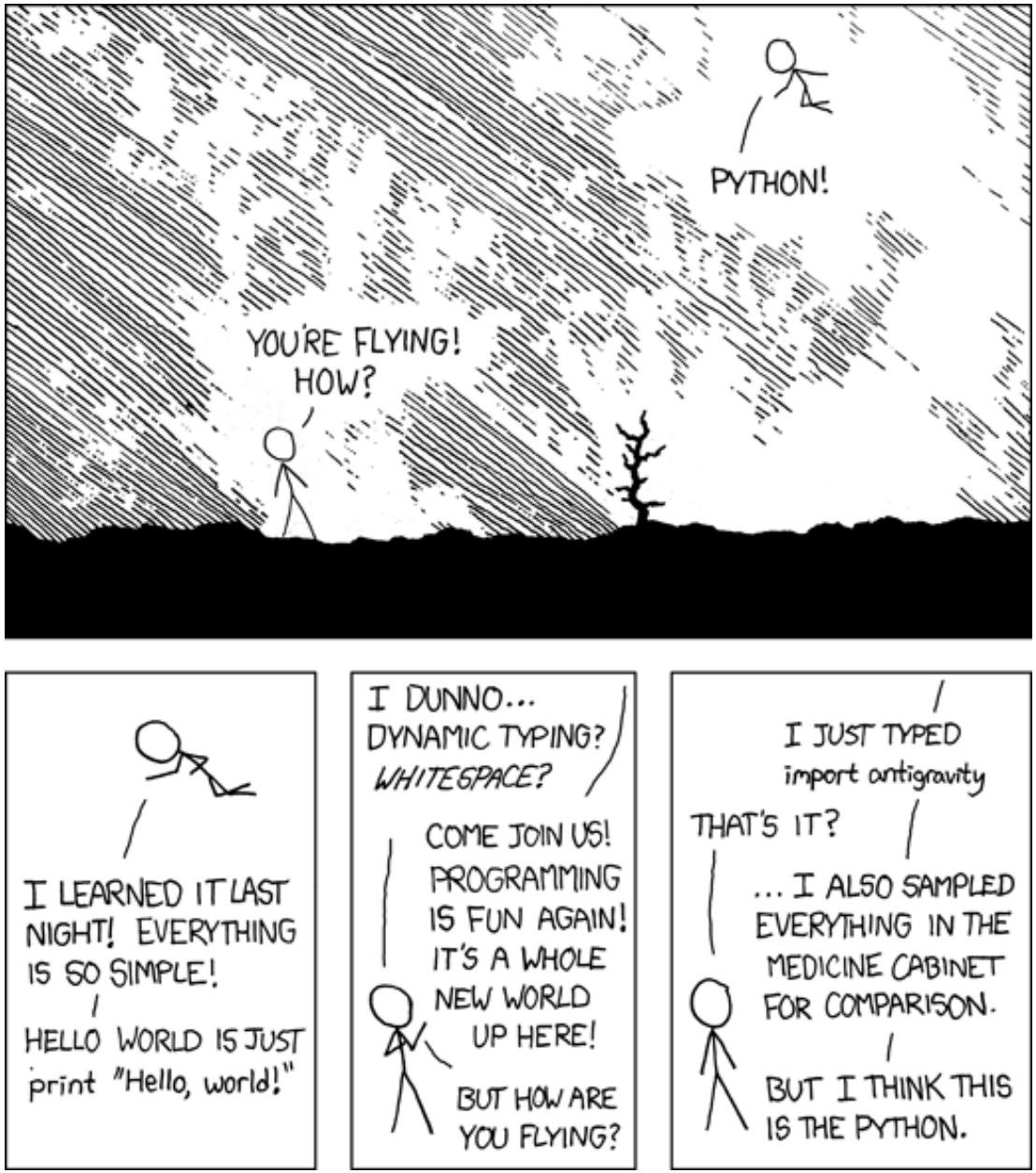
Goals: Why we started project

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

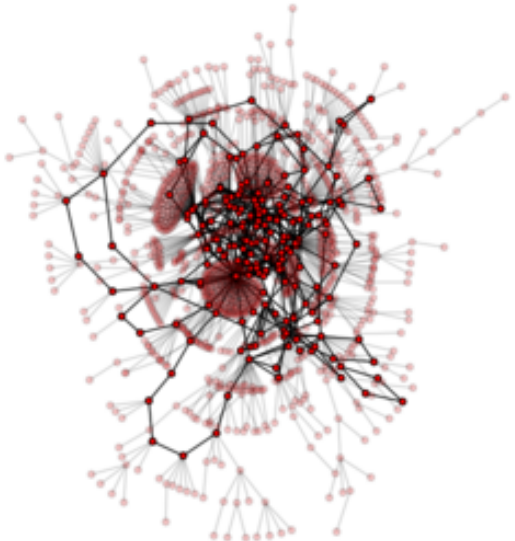
- No existing API or graph implementation that was suitable
- Inspired by Guido van Rossum's 1998 Python graph representation essay
- First public release in April 2005

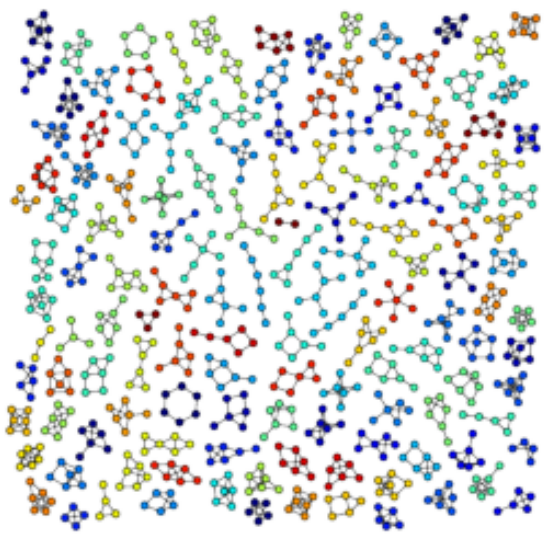
Also: Fun


YOU'RE FLYING! HOW?
PYTHON!
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

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
- Flexibility ideal for representing networks found in many different fields
- Many unit and functional tests
- Online up-to-date documentation





Design decisions

NetworkX defines no custom node objects or edge objects

- “node-centric” view of network
- Nodes: whatever you put in (hashable)
- Edges: tuples with optional edge data (stored in dictionary)
- Edge data is arbitrary and users can define custom node types

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

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

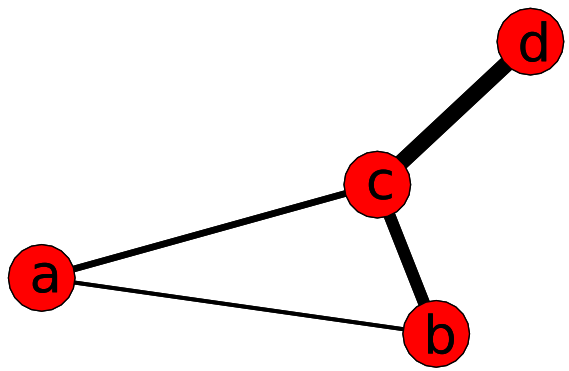
```
>>> G.add_edge(1,'a')
>>> G.add_edge('b',math.cos)
>>> print G.edges()
[('b', <built-in function cos>), ('a', 1)]
```

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

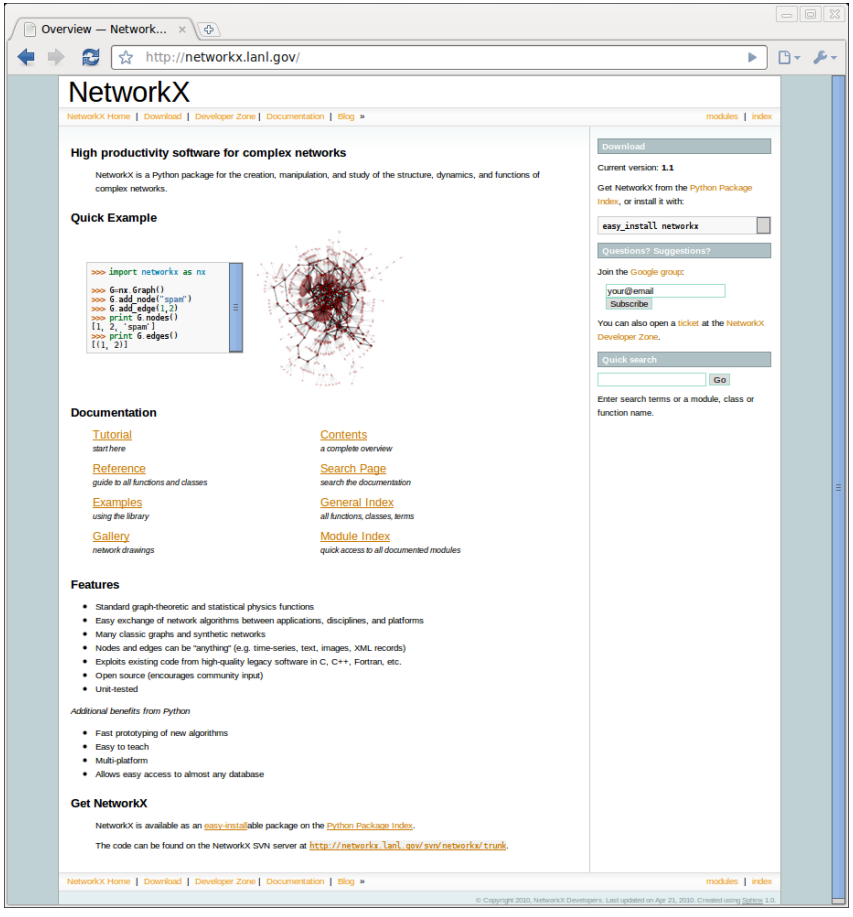
NetworkX has many tests that can be run by users

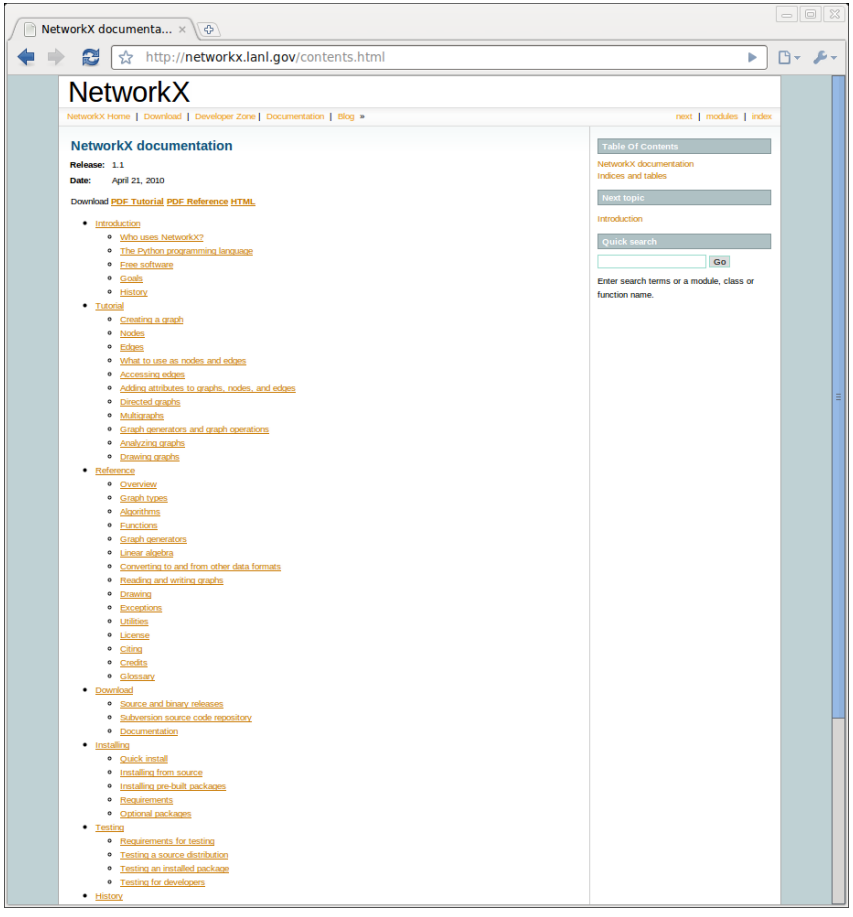
```
>>> import networkx
>>> networkx.test(verbosity=2)
...
Doctest: networkx.utils ... ok
Conversion from digraph to array to digraph. ... ok
Conversion from digraph to matrix to digraph. ... ok
Conversion from graph to array to graph. ... ok
Conversion from graph to matrix to graph. ... ok
Conversion 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 855 tests in 4.334s

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

```
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(with_labels=True),delta_in)
        elif r < alpha+beta:
            v = _choose_node(G, G.out_degree(with_labels=True),delta_out)
            w = _choose_node(G, G.in_degree(with_labels=True),delta_in)
        else:
            v = _choose_node(G, G.out_degree(with_labels=True),delta_out)
            w = len(G)
        G.add_edge(v,w)
    return G
```

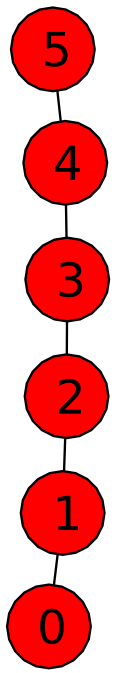
Feature: leveraging libraries

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]
```

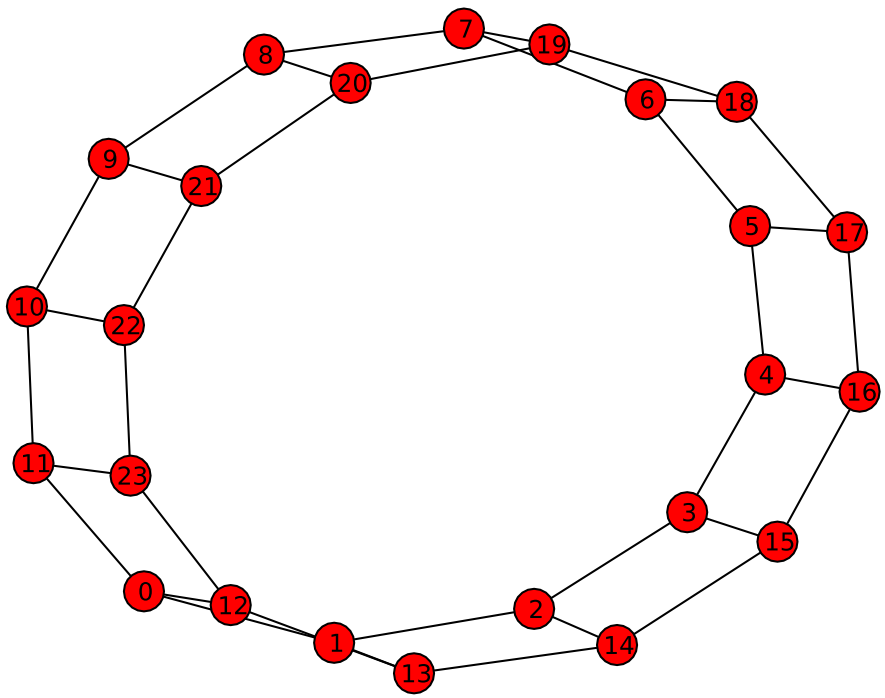


Feature: drawing

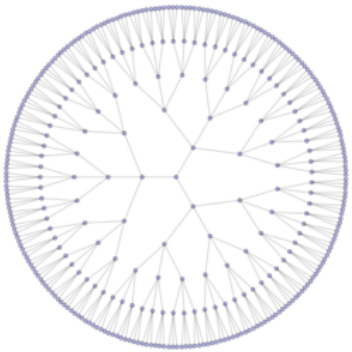
Built-in interface to Matplotlib plotting package

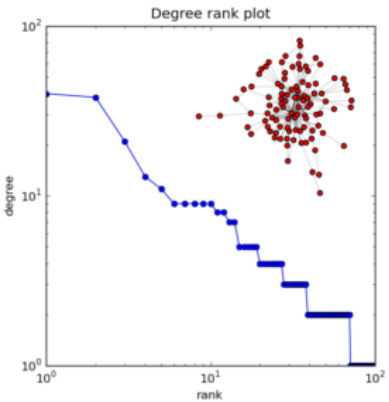
Node positioning algorithms based on force-directed, spectral, and geometric methods

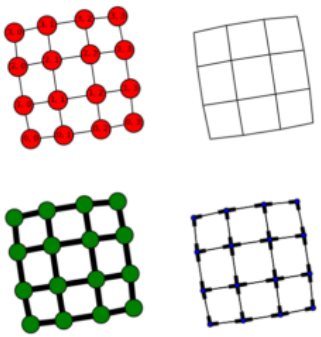
```
>>> G = nx.circular_ladder_graph(12)
>>> nx.draw(G) # Matplotlib under the hood
```

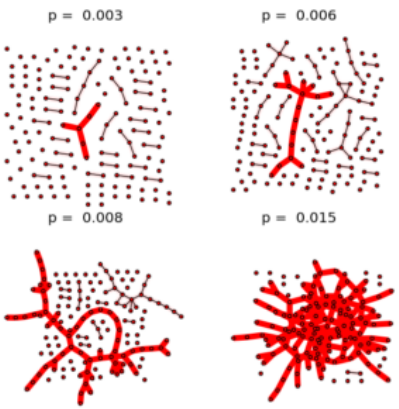


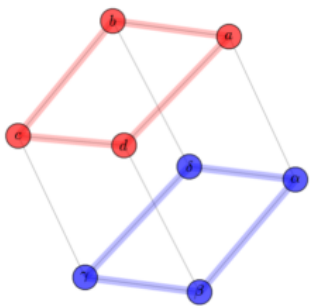
Drawing with Matplotlib

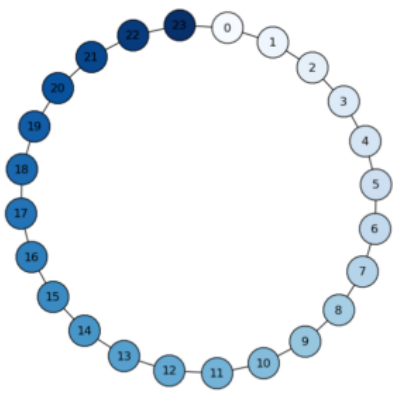


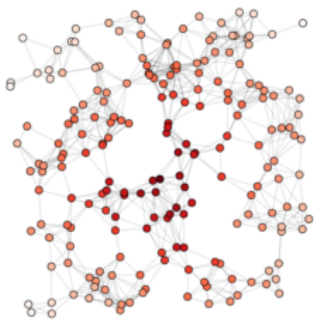






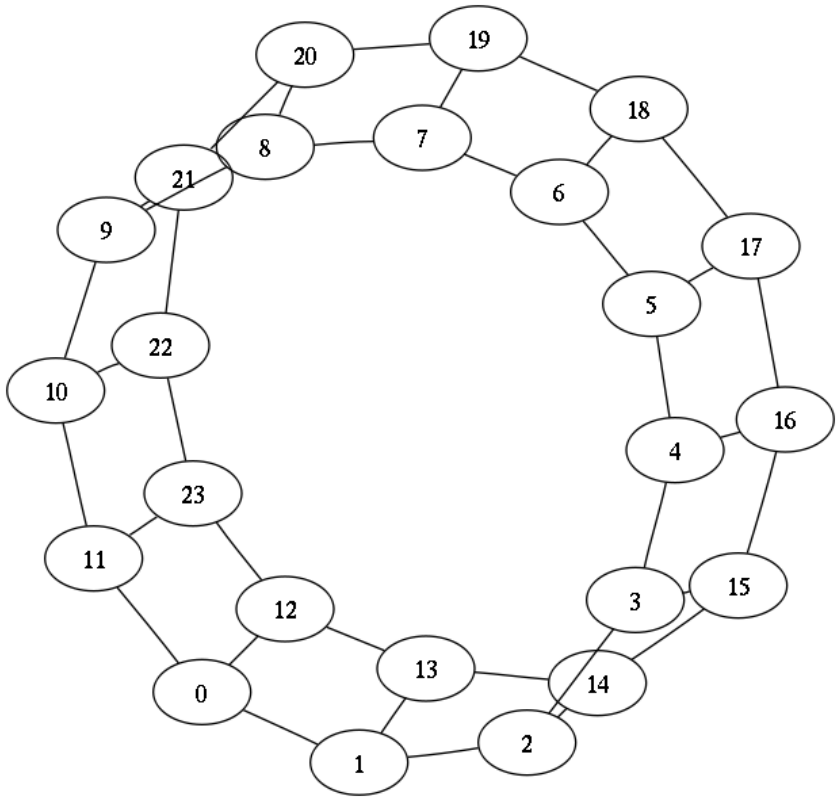






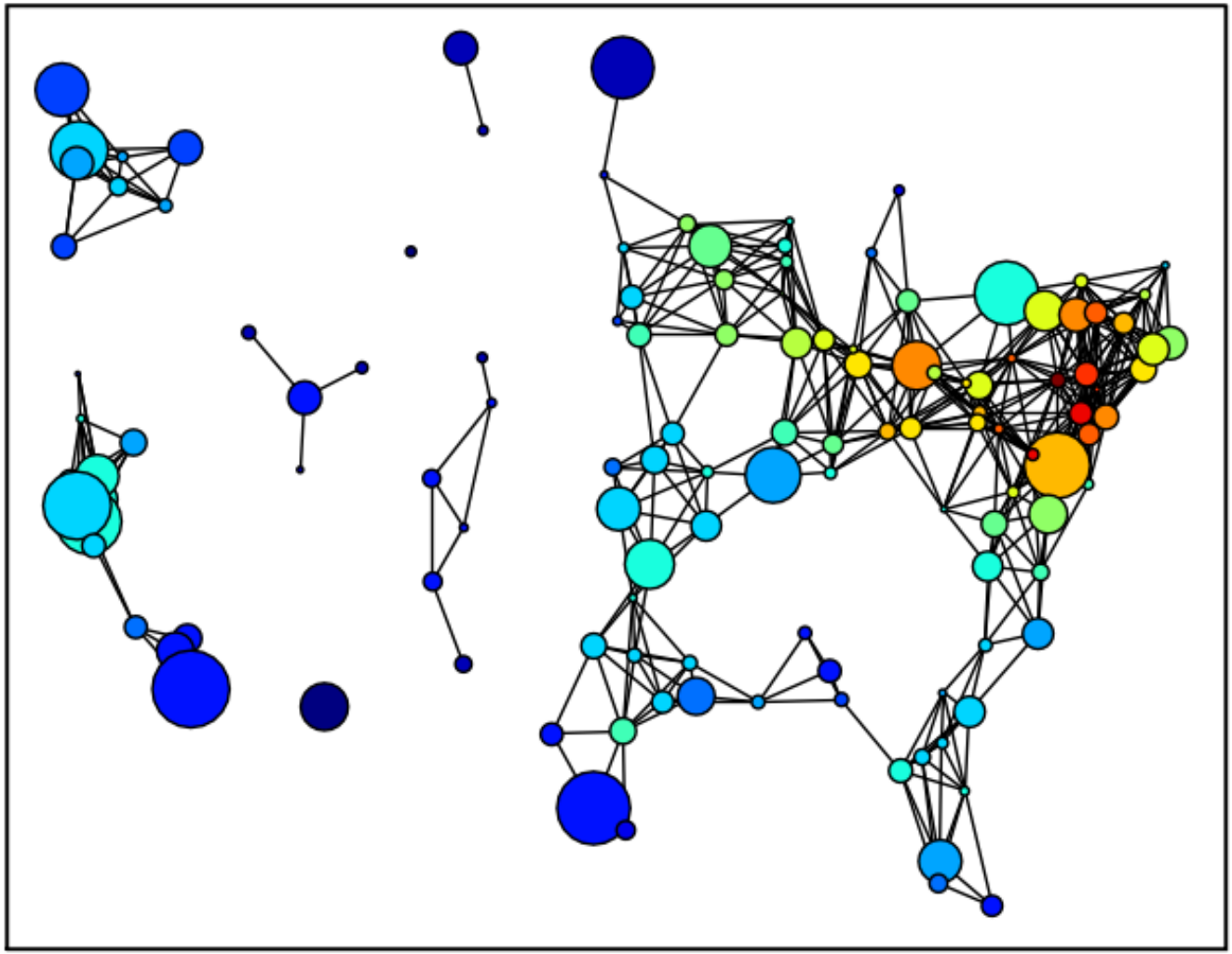
Drawing with other programs

Graphviz



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

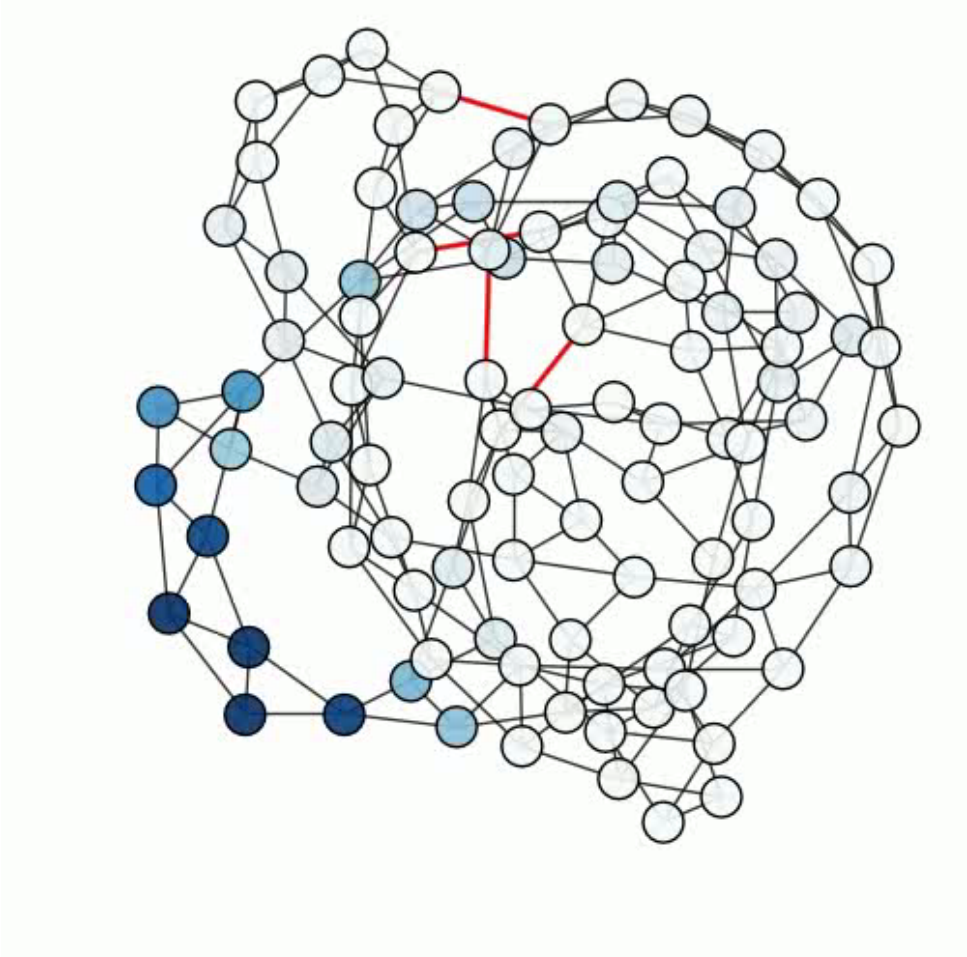
Where is NetworkX being used?



Los Alamos: Synchronization of networks of oscillators

Adding red edges allows network to synchronize.

Edges found by studying network Laplacian spectrum.

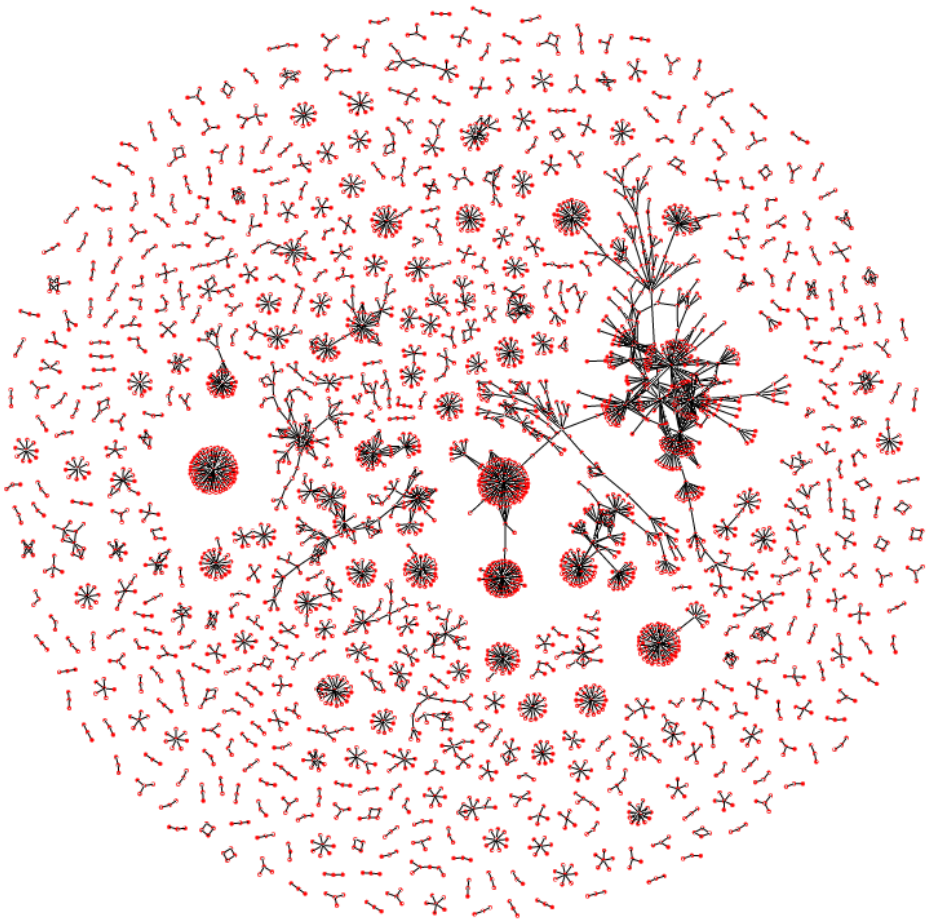


Aric Hagberg and Daniel A. Schult, "Rewiring networks for synchronization", Chaos, 2008

Cornell: Education

I

NFO/SOCI 485 Computational Methods for Complex Networks
(Gueorgi Kossinets)
Physics 7682 / Computing & Information Sciences 6229
(Chris Myers)

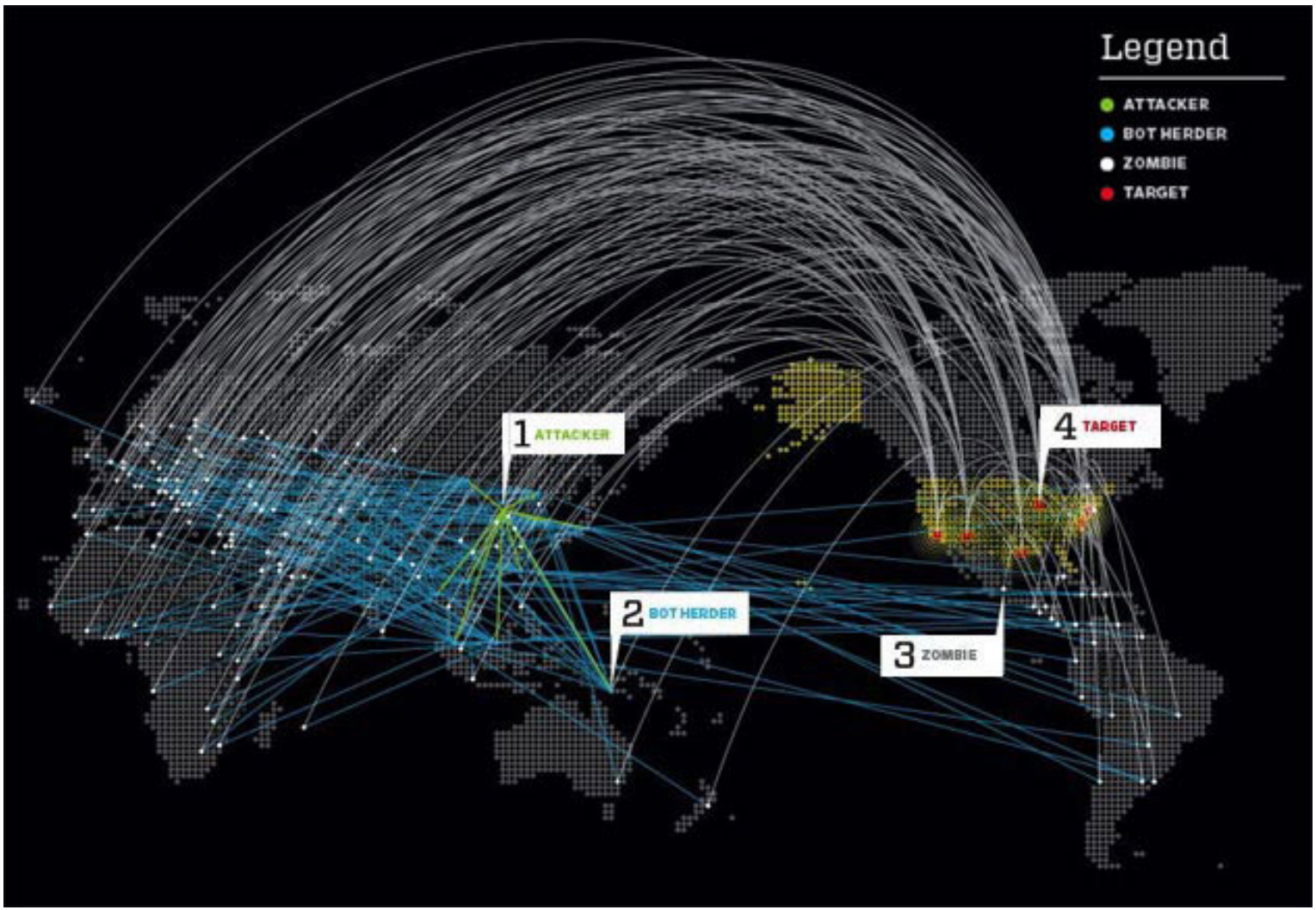


"Reality mining" (Nathan Eagle)

Inferring social network structure using cell phone data
e.g. 1.2M phone users in Rwanda

FBI: Cybercrime

Fighting cybercrime: botnets, spam, phishing

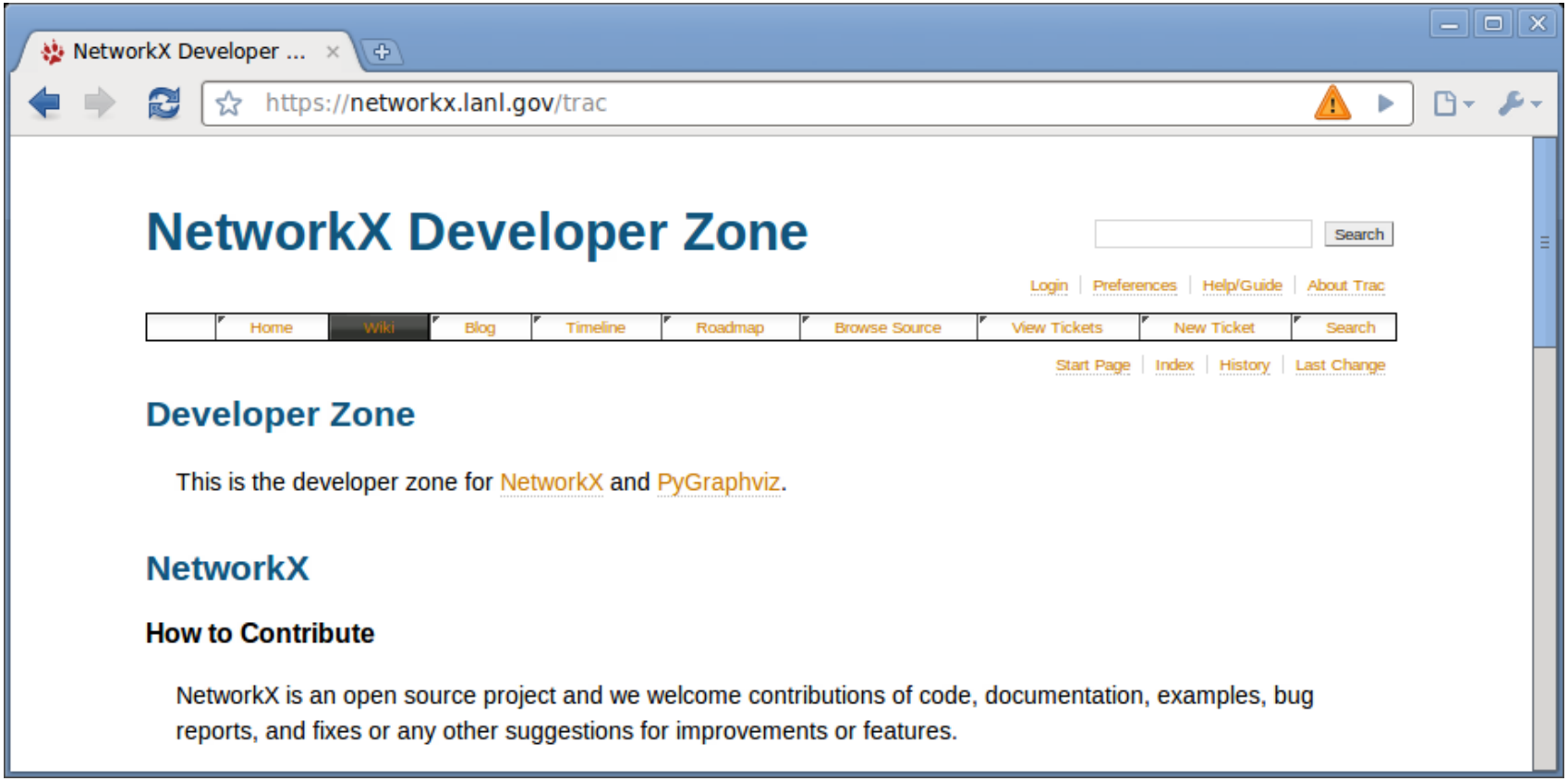


Current status

`http://networkx.lanl.gov/`

Currently at version networkx-1.1

Active development: community driven, community supported project.



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