

A basic introduction to Complex Networks, using python

Marcelo Ponce



April 4th, 2016



Outline

1 Introduction

- Motivation & Examples

2 Basic Elements

- Python Packages
- Building your own CN: **NetworkX**
- Networks
- Several types of graphs/networks in **NetworkX**
- Basic Network Analysis

3 Real-life applications

- Twitter relationships
- SciNet GPC cluster

4 Simulating CNs

- PyCX

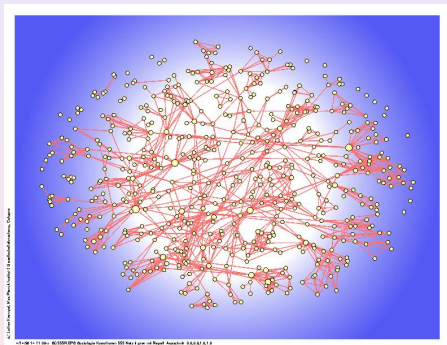
5 References

Examples of Complex Networks

A system composed of many nonlinear interacting units often forms a complex system with new emergent properties that are not held by the individual units.

Examples:

- circadian rhythms,
- electrochemical reactions,
- laser arrays,
- neuron networks,
- populations of fireflies,
- Josephson junction arrays...,
etc.



▲ Co-Authorship Network By Lothar Krempel

<http://www.mpi-fg->

[koeln.mpg.de/~lk/netvis/Huge.html](http://www.mpi-fg-koeln.mpg.de/~lk/netvis/Huge.html)

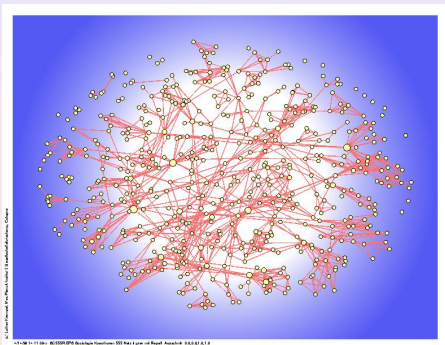
http://www.nd.edu/~networks/Image_Gallery/gallery.html

Examples of Complex Networks

A system composed of many nonlinear interacting units often forms a complex system with new emergent properties that are not held by the individual units.

Examples:

- circadian rhythms,
- electrochemical reactions,
- laser arrays,
- neuron networks,
- populations of fireflies,
- Josephson junction arrays..., etc.



▲ Co-Authorship Network By Lothar Krempel

<http://www.mpi-fg->

[koeln.mpg.de/~lk/netvis/Huge.html](http://www.mpi-fg-koeln.mpg.de/~lk/netvis/Huge.html)

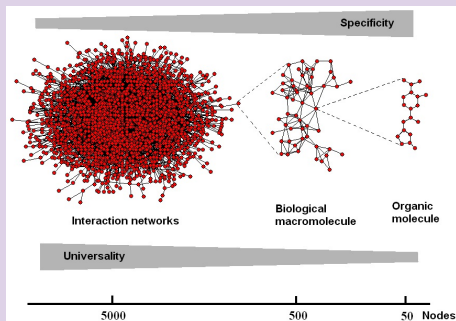
http://www.nd.edu/~networks/Image_Gallery/gallery.html

Examples of Complex Networks

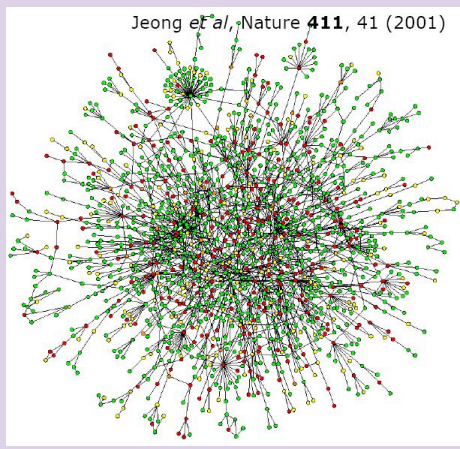
Biological networks...

▼ Proteins, molecules, chemical reactions...

▼ Genes, proteins, ...



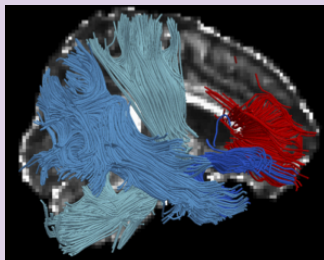
Map of protein-protein interactions. The colour of a node signifies the phenotypic effect of removing the corresponding protein (lethal; non-lethal; slow growth; unknown). ▶



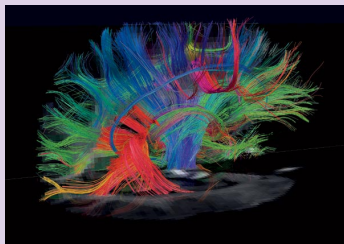
Examples of Complex Networks

Brain networks...

▼ Physiological networks



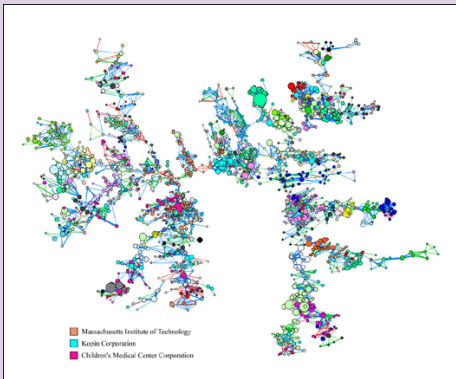
▼ Neurological pathways



Examples of Complex Networks

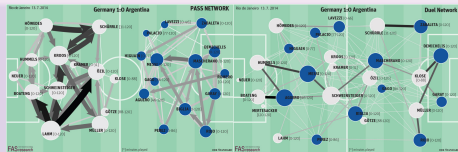
Social networks...

▼ Boston Inventor Networks mid-1990s



<http://derstandard.at/2000002589338/Messi-glaentz-als-anwesend-Abwesender>

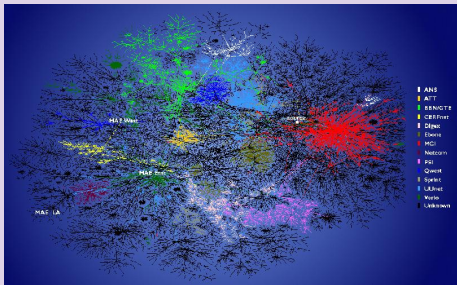
▼ Sports: Futbol - world cup...



Examples of Complex Networks

Internet

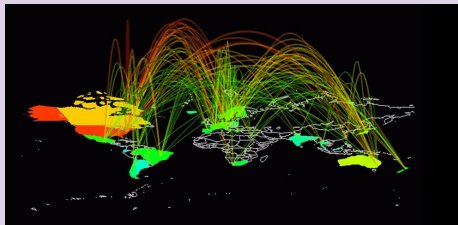
▼ Internet connectivity,



with selected backbone ISPs (Internet Service Provider)

colored separately

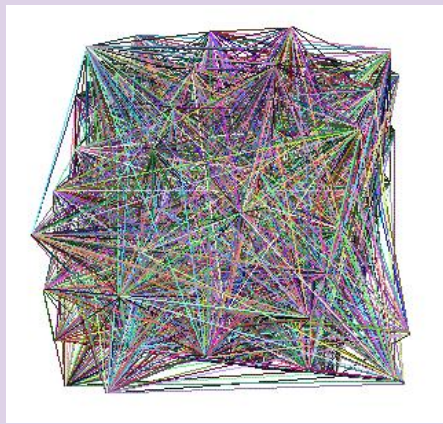
▼ Arc map showing the world-wide internet traffic



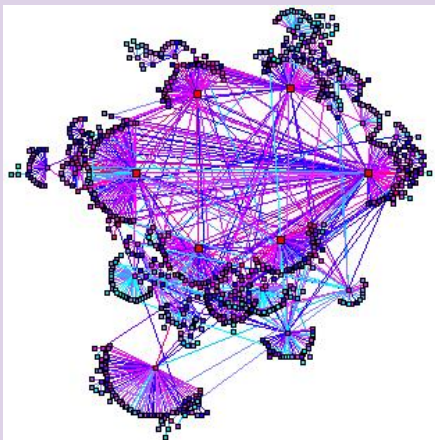
Examples of Complex Networks

Internet

▼ Mapping the internet: tracing domain routes



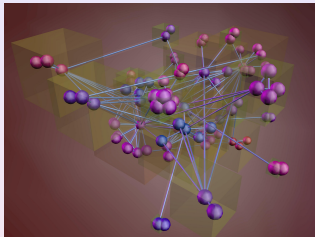
▼ Hierarchical topology of the international web cache



nodes, network, maps...

A **network** is a set of **nodes** interconnected via **links**.

- 1 [nodes]: ensemble of N oscillators
- 2 [dynamics+network]: each oscillator evolves according to its own **intrinsic dynamics** [map (fn)] + a **coupling term** [network].
- 3 The coupling term is defined by:
 - **coupling strength**
 - **coupling topology** (all-to-all, n-n, etc)



► **Nodes** and **Links** can be anything depending on the context

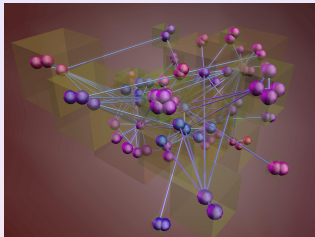
▼ from the previous examples:

- **Internet**: **Nodes** \rightsquigarrow routers & **Links** \rightsquigarrow optical fibers.
- **WWW**: **Nodes** \rightsquigarrow document files & **Links** \rightsquigarrow hyperlinks.
- **The Scientific Citation Network**: **Nodes** \rightsquigarrow papers & **Links** \rightsquigarrow citations.
- **Social Networks**: **Nodes** \rightsquigarrow individuals & **Links** \rightsquigarrow relations.

nodes, network, maps...

A **network** is a set of **nodes** interconnected via **links**.

- 1 [nodes]: ensemble of N oscillators
- 2 [dynamics+network]: each oscillator **evolves** according to its own **intrinsic dynamics** [map (fn)] + a **coupling term** [network].
- 3 The **coupling term** is defined by:
 - **coupling strength**
 - **coupling topology** (all-to-all, n-n, etc)



► Nodes and Links can be anything depending on the context

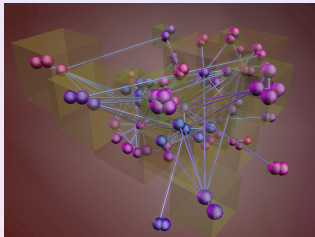
▼ from the previous examples:

- Internet: Nodes \rightsquigarrow routers & Links \rightsquigarrow optical fibers.
- WWW: Nodes \rightsquigarrow document files & Links \rightsquigarrow hyperlinks.
- The Scientific Citation Network: Nodes \rightsquigarrow papers & Links \rightsquigarrow citations.
- Social Networks: Nodes \rightsquigarrow individuals & Links \rightsquigarrow relations.

nodes, network, maps...

A **network** is a set of **nodes** interconnected via **links**.

- 1 [nodes]: ensemble of N oscillators
- 2 [dynamics+network]: each oscillator **evolves** according to its own **intrinsic dynamics** [map (fn)] + a **coupling term** [network].
- 3 The **coupling term** is defined by:
 - **coupling strength**
 - **coupling topology** (all-to-all, n-n, etc)



▶ **Nodes** and **Links** can be anything depending on the **context**

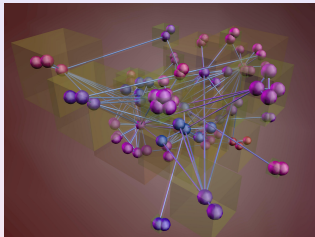
▼ from the previous examples:

- ▶ **Internet**: **Nodes** \rightsquigarrow routers & **Links** \rightsquigarrow optical fibers.
- ▶ **WWW**: **Nodes** \rightsquigarrow document files & **Links** \rightsquigarrow hyperlinks.
- ▶ **The Scientific Citation Network**: **Nodes** \rightsquigarrow papers & **Links** \rightsquigarrow citations.
- ▶ **Social Networks**: **Nodes** \rightsquigarrow individuals & **Links** \rightsquigarrow relations.

nodes, network, maps...

A **network** is a set of **nodes** interconnected via **links**.

- 1 [nodes]: ensemble of N oscillators
- 2 [dynamics+network]: each oscillator **evolves** according to its own **intrinsic dynamics** [map (fn)] + a **coupling term** [network].
- 3 The **coupling term** is defined by:
 - **coupling strength**
 - **coupling topology** (all-to-all, n-n, etc)



▶ **Nodes** and **Links** can be anything depending on the **context**

▼ from the previous examples:

- ▶ **Internet**: **Nodes** \rightsquigarrow routers & **Links** \rightsquigarrow optical fibers.
- ▶ **WWW**: **Nodes** \rightsquigarrow document files & **Links** \rightsquigarrow hyperlinks.
- ▶ **The Scientific Citation Network**: **Nodes** \rightsquigarrow papers & **Links** \rightsquigarrow citations.
- ▶ **Social Networks**: **Nodes** \rightsquigarrow individuals & **Links** \rightsquigarrow relations.

CN: several packages & tools available in Python

- **networkX**: <http://networkx.lanl.gov/>
- **PyCX Project**: <http://pycx.sf.net/>
- ComplexNetworkSim
- SimPy
- graph-tool [<http://graph-tool.skewed.de/>]
- pyGraphViz (GraphViz) [<http://www.graphviz.org/>]
- igraph [<http://igraph.org/>]
- python-graph
- gephi [<http://gephi.org/>]
- ...

CN: several packages & tools available in Python

- **networkX**: <http://networkx.lanl.gov/>
- **PyCX Project**: <http://pycx.sf.net/>
- **ComplexNetworkSim**
- **SimPy**
- **graph-tool** [<http://graph-tool.skewed.de/>]
- **pyGraphViz (GraphViz)** [<http://www.graphviz.org/>]
- **igraph** [<http://igraph.org/>]
- **python-graph**
- **gephi** [<http://gephi.org/>]
- ...

Required Packages



www.scipy.org



www.numpy.org



matplotlib.org

▾ NetworkX

networkx.github.io

networkx.lanl.gov

▸ pyGraphViz

▾ PyCX

pycx.sf.net

▸ TKinter

Required Packages



www.scipy.org



www.numpy.org



matplotlib.org

▼ NetworkX

networkx.github.io
networkx.lanl.gov

▶ pyGraphViz

▼ PyCX

pycx.sf.net

▶ TKinter

Required Packages



www.scipy.org



www.numpy.org



matplotlib.org

▾ NetworkX

networkx.github.io
networkx.lanl.gov

▸ pyGraphViz

▾ PyCX

pycx.sf.net

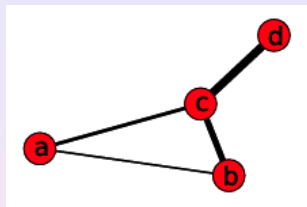
▸ TKinter

Defining a graph

```
import networkx as nx

g = nx.Graph()

g.add_edge('a', 'b', weight=0.1)
g.add_edge('b', 'c', weight=1.5)
g.add_edge('a', 'c', weight=1.0)
g.add_edge('c', 'd', weight=2.2)
```



Some basic props. of the graph

```
g.number_of_nodes() # also g.order()
4
g.order() # also g.size()
4
```

```
g.nodes()
['a', 'c', 'b', 'd']
g.edges()
[('a', 'c'), ('a', 'b'), ('c', 'b'), ('c', 'd')]
g.edges('b')
[('b', 'a'), ('b', 'c')]
```

```
g.degree('a')
2
g.neighbors('c')
['a', 'b', 'd']
```

```
print nx.shortest_path(g, 'b', 'd')
['b', 'c', 'd']
```

```
print nx.shortest_path(g, 'b', 'd', weight='weight')
['b', 'a', 'c', 'd']
```

```
import pylab as plt
plt.ion()
nx.draw(g)
plt.show()
```

Defining a graph

```
import networkx as nx

g = nx.Graph()

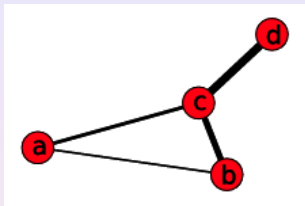
g.add_edge('a', 'b', weight=0.1)
g.add_edge('b', 'c', weight=1.5)
g.add_edge('a', 'c', weight=1.0)
g.add_edge('c', 'd', weight=2.2)
```

Some basic props. of the graph

```
g.number_of_nodes() # also g.order()
4
g.order() # also g.size()
4
```

```
g.nodes()
['a', 'c', 'b', 'd']
g.edges()
[('a', 'c'), ('a', 'b'), ('c', 'b'), ('c', 'd')]
g.edges('b')
[('b', 'a'), ('b', 'c')]
```

```
g.degree('a')
2
g.neighbors('c')
['a', 'b', 'd']
```



```
print nx.shortest_path(g, 'b', 'd')
['b', 'c', 'd']
```

```
print nx.shortest_path(g, 'b', 'd', weight='weight')
['b', 'a', 'c', 'd']
```

```
import pylab as plt
plt.ion()
nx.draw(g)
plt.show()
```


Defining a graph

```
import networkx as nx
```

```
g = nx.Graph()
```

```
g.add_edge('a', 'b', weight=0.1)
```

```
g.add_edge('b', 'c', weight=1.5)
```

```
g.add_edge('a', 'c', weight=1.0)
```

```
g.add_edge('c', 'd', weight=2.2)
```

Some basic props. of the graph

```
g.number_of_nodes() # also g.order()
```

```
4
```

```
g.order() # also g.size()
```

```
4
```

```
g.nodes()
```

```
['a', 'c', 'b', 'd']
```

```
g.edges()
```

```
[('a', 'c'), ('a', 'b'), ('c', 'b'), ('c', 'd')]
```

```
g.edges('b')
```

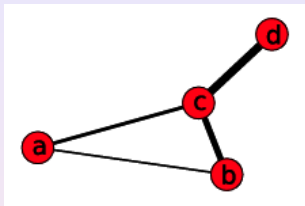
```
[('b', 'a'), ('b', 'c')]
```

```
g.degree('a')
```

```
2
```

```
g.neighbors('c')
```

```
['a', 'b', 'd']
```



```
print nx.shortest_path(g, 'b', 'd')
['b', 'c', 'd']
```

```
print nx.shortest_path(g, 'b', 'd', weight='weight')
['b', 'a', 'c', 'd']
```

```
import pylab as plt
plt.ion()
nx.draw(g)
plt.show()
```

Defining a graph

```
import networkx as nx
```

```
g = nx.Graph()
```

```
g.add_edge('a', 'b', weight=0.1)
```

```
g.add_edge('b', 'c', weight=1.5)
```

```
g.add_edge('a', 'c', weight=1.0)
```

```
g.add_edge('c', 'd', weight=2.2)
```

Some basic props. of the graph

```
g.number_of_nodes() # also g.order()
```

```
4
```

```
g.order() # also g.size()
```

```
4
```

```
g.nodes()
```

```
['a', 'c', 'b', 'd']
```

```
g.edges()
```

```
[('a', 'c'), ('a', 'b'), ('c', 'b'), ('c', 'd')]
```

```
g.edges('b')
```

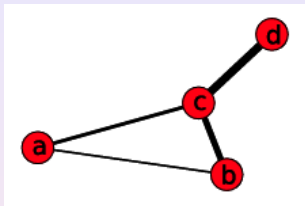
```
[('b', 'a'), ('b', 'c')]
```

```
g.degree('a')
```

```
2
```

```
g.neighbors('c')
```

```
['a', 'b', 'd']
```



```
print nx.shortest_path(g, 'b', 'd')
['b', 'c', 'd']
```

```
print nx.shortest_path(g, 'b', 'd', weight='weight')
['b', 'a', 'c', 'd']
```

```
import pylab as plt
plt.ion()
nx.draw(g)
plt.show()
```

Defining a graph

```
import networkx as nx
```

```
g = nx.Graph()
```

```
g.add_edge('a', 'b', weight=0.1)
```

```
g.add_edge('b', 'c', weight=1.5)
```

```
g.add_edge('a', 'c', weight=1.0)
```

```
g.add_edge('c', 'd', weight=2.2)
```

Some basic props. of the graph

```
g.number_of_nodes() # also g.order()
```

```
4
```

```
g.order() # also g.size()
```

```
4
```

```
g.nodes()
```

```
['a', 'c', 'b', 'd']
```

```
g.edges()
```

```
[('a', 'c'), ('a', 'b'), ('c', 'b'), ('c', 'd')]
```

```
g.edges('b')
```

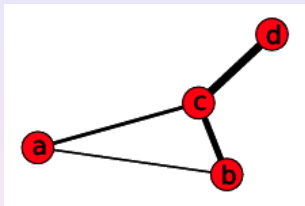
```
[('b', 'a'), ('b', 'c')]
```

```
g.degree('a')
```

```
2
```

```
g.neighbors('c')
```

```
['a', 'b', 'd']
```



```
print nx.shortest_path(g, 'b', 'd')
['b', 'c', 'd']
```

```
print nx.shortest_path(g, 'b', 'd', weight='weight')
['b', 'a', 'c', 'd']
```

```
import pylab as plt
plt.ion()
nx.draw(g)
plt.show()
```

Defining a graph

```
import networkx as nx
```

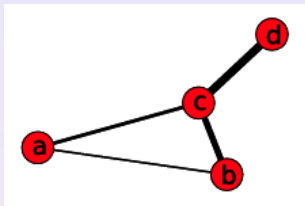
```
g = nx.Graph()
```

```
g.add_edge('a', 'b', weight=0.1)
```

```
g.add_edge('b', 'c', weight=1.5)
```

```
g.add_edge('a', 'c', weight=1.0)
```

```
g.add_edge('c', 'd', weight=2.2)
```



Some basic props. of the graph

```
g.number_of_nodes() # also g.order()
```

```
4
```

```
g.order() # also g.size()
```

```
4
```

```
g.nodes()
```

```
['a', 'c', 'b', 'd']
```

```
g.edges()
```

```
[('a', 'c'), ('a', 'b'), ('c', 'b'), ('c', 'd')]
```

```
g.edges('b')
```

```
[('b', 'a'), ('b', 'c')]
```

```
g.degree('a')
```

```
2
```

```
g.neighbors('c')
```

```
['a', 'b', 'd']
```

```
print nx.shortest_path(g, 'b', 'd')
['b', 'c', 'd']
```

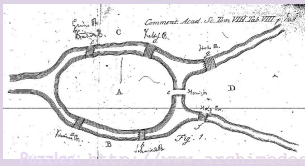
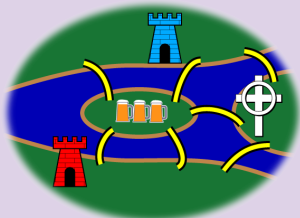
```
print nx.shortest_path(g, 'b', 'd', weight='weight')
['b', 'a', 'c', 'd']
```

```
import pylab as plt
plt.ion()
nx.draw(g)
plt.show()
```

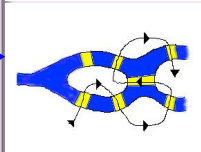
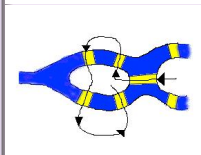
Network Topology

Graph Theory

7 bridges of Königsberg [Euler, 1736]

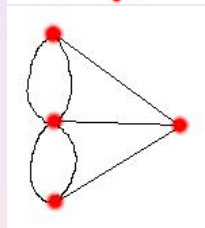
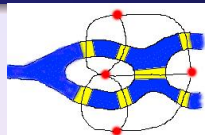
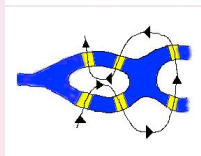


7 bridges



NO soln!!!

6 bridges

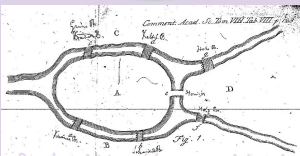
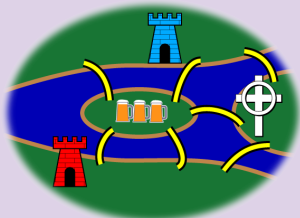


"If a network has more than two odd vertices, it does NOT have an Euler path."

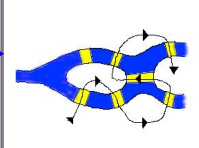
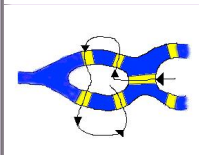
Network Topology

Graph Theory

7 bridges of Königsberg [Euler, 1736]

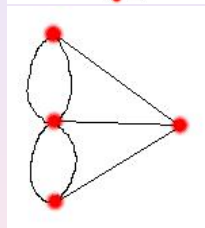
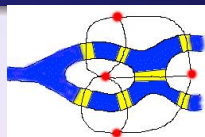
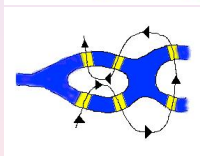


7 bridges



▶ NO soln!!!

6 bridges

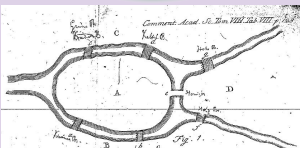
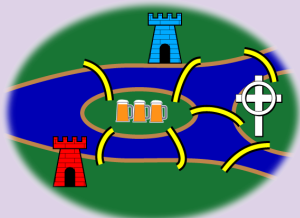
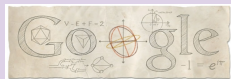


“If a network has more than two odd vertices, it does NOT have an Euler path.”

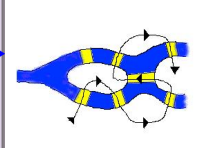
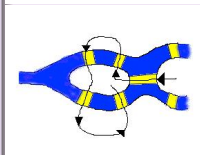
Network Topology

Graph Theory

7 bridges of Königsberg [Euler, 1736]

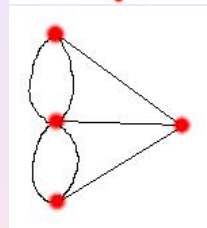
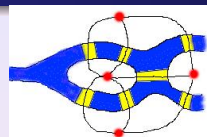
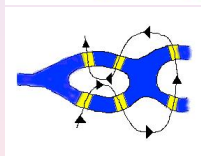


7 bridges



▶ NO soln!!!

6 bridges

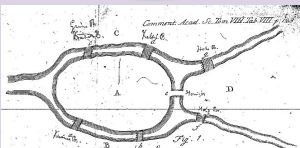
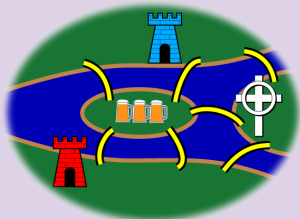


“If a network has more than two odd vertices, it does NOT have an Euler path.”

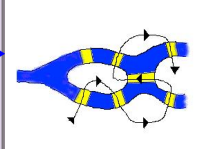
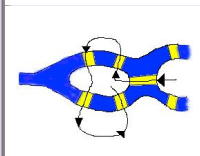
Network Topology

Graph Theory

- 7 bridges of Königsberg [Euler, 1736]

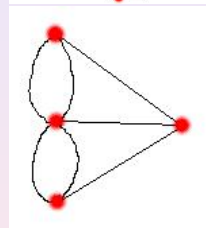
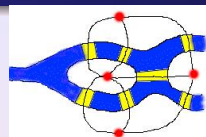
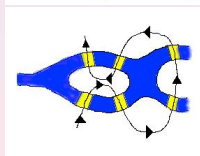


- 7 bridges



- NO soln!!!

- 6 bridges

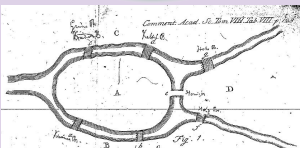
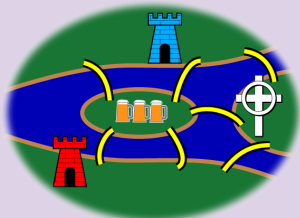
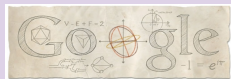


“If a network has more than two odd vertices, it does NOT have an Euler path.”

Network Topology

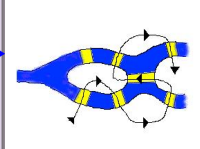
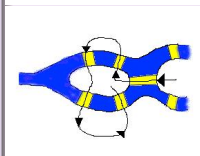
Graph Theory

7 bridges of Königsberg [Euler, 1736]



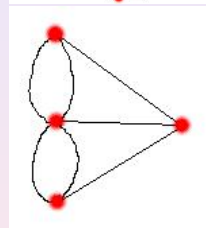
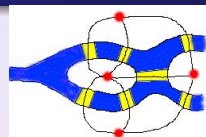
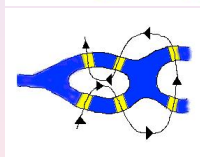
Puzzles: http://www.archimedes-lab.org/How_to_Solve/Water_gas.html
<http://www.mathsisfun.com/activity/seven-bridges-koenigsberg.html>

7 bridges



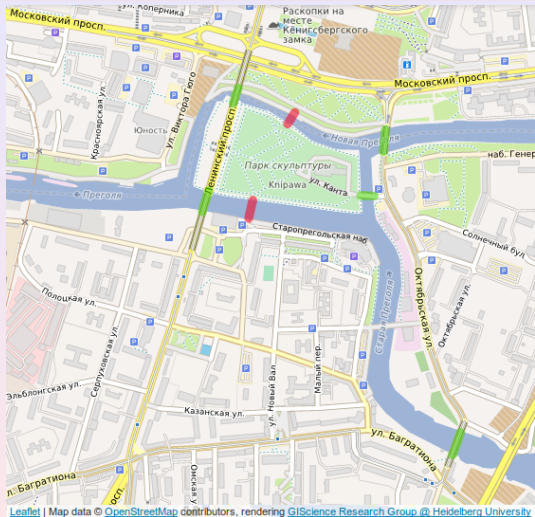
▶ NO soln!!!

6 bridges



“If a network has more than two odd vertices, it does NOT have an Euler path.”

Present state of the seven bridges of Königsberg



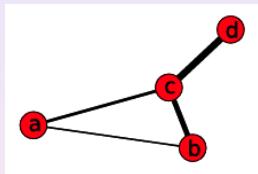
Src:

http://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg

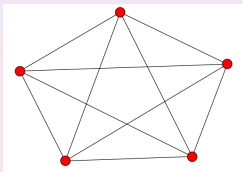
Eulerian path/circuit with NetworkX

How to solve this problem with `python-NetworkX`?

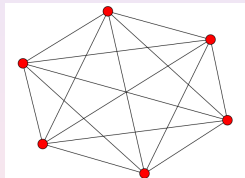
⇒ Check if the graph holds a *Eulerian path*: `is_eulerian()`



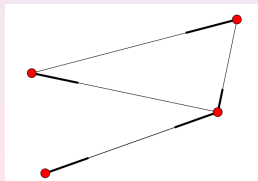
```
nx.is_eulerian(g)
False
```



```
g5 = nx.complete_graph(5)
nx.draw(g5)
nx.is_eulerian(g5)
True
```



```
g6 = nx.complete_graph(6)
nx.draw(g6)
nx.is_eulerian(g6)
False
```

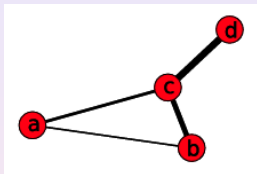


```
h = nx.DiGraph([0:[3], 1:[2], 2:[3], 3:[0, 1]])
nx.draw(h)
nx.is_eulerian(h)
True
```

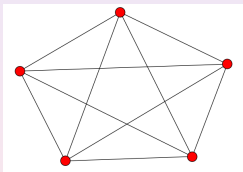
Eulerian path/circuit with NetworkX

How to solve this problem with `python-NetworkX`?

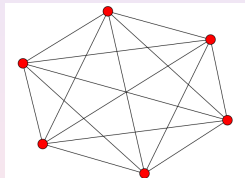
► Check if the graph holds a *Eulerian path*: `is_eulerian()`



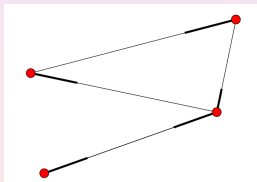
```
nx.is_eulerian(g)
False
```



```
g5 = nx.complete_graph(5)
nx.draw(g5)
nx.is_eulerian(g5)
True
```



```
g6 = nx.complete_graph(6)
nx.draw(g6)
nx.is_eulerian(g6)
False
```

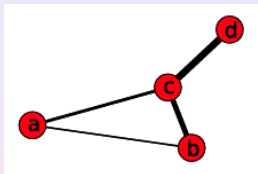


```
h = nx.DiGraph([0:[3], 1:[2], 2:[3], 3:[0, 1]])
nx.draw(h)
nx.is_eulerian(h)
True
```

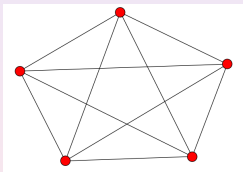
Eulerian path/circuit with NetworkX

How to solve this problem with `python-NetworkX`?

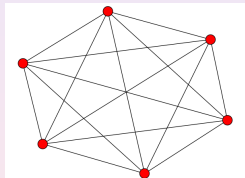
► Check if the graph holds a *Eulerian path*: `is_eulerian()`



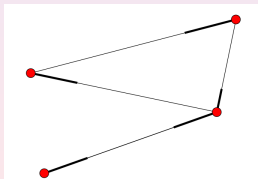
```
nx.is_eulerian(g)
False
```



```
g5 = nx.complete_graph(5)
nx.draw(g5)
nx.is_eulerian(g5)
True
```



```
g6 = nx.complete_graph(6)
nx.draw(g6)
nx.is_eulerian(g6)
False
```

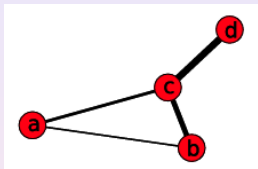


```
h=nx.DiGraph([(0:[3], 1:[2], 2:[3], 3:[0, 1])])
nx.draw(h)
nx.is_eulerian(h)
True
```

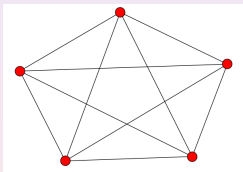
Eulerian path/circuit with NetworkX

How to solve this problem with `python-NetworkX`?

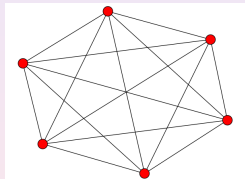
► Check if the graph holds a *Eulerian path*: `is_eulerian()`



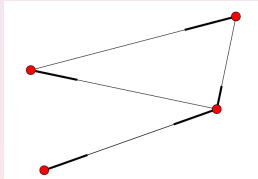
```
nx.is_eulerian(g)
False
```



```
g5 = nx.complete_graph(5)
nx.draw(g5)
nx.is_eulerian(g5)
True
```



```
g6 = nx.complete_graph(6)
nx.draw(g6)
nx.is_eulerian(g6)
False
```

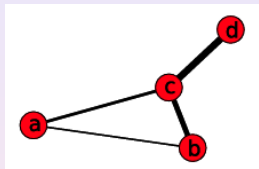


```
h = nx.DiGraph([0:[3], 1:[2], 2:[3], 3:[0, 1]])
nx.draw(h)
nx.is_eulerian(h)
True
```

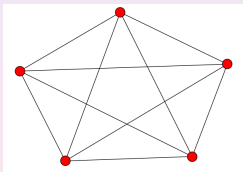
Eulerian path/circuit with NetworkX

How to solve this problem with `python-NetworkX`?

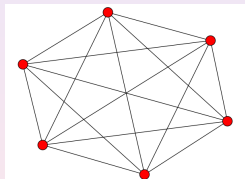
► Check if the graph holds a *Eulerian path*: `is_eulerian()`



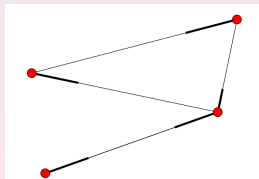
```
nx.is_eulerian(g)
False
```



```
g5 = nx.complete_graph(5)
nx.draw(g5)
nx.is_eulerian(g5)
True
```



```
g6 = nx.complete_graph(6)
nx.draw(g6)
nx.is_eulerian(g6)
False
```

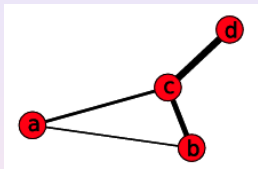


```
h=nx.DiGraph([(0:[3], 1:[2], 2:[3], 3:[0, 1])])
nx.draw(h)
nx.is_eulerian(h)
True
```

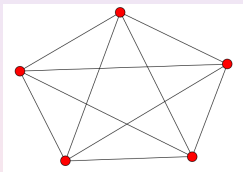
Eulerian path/circuit with NetworkX

How to solve this problem with `python-NetworkX`?

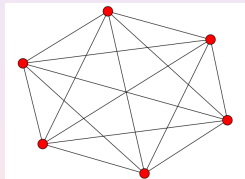
► Check if the graph holds a *Eulerian path*: `is_eulerian()`



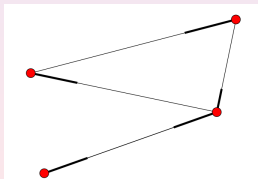
```
nx.is_eulerian(g)
False
```



```
g5 = nx.complete_graph(5)
nx.draw(g5)
nx.is_eulerian(g5)
True
```



```
g6 = nx.complete_graph(6)
nx.draw(g6)
nx.is_eulerian(g6)
False
```

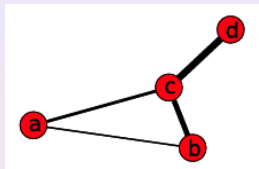


```
h=nx.DiGraph([(0:[3], 1:[2], 2:[3], 3:[0, 1])])
nx.draw(h)
nx.is_eulerian(h)
True
```

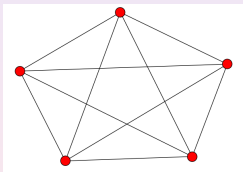

Eulerian path/circuit with NetworkX

How to solve this problem with `python-NetworkX`?

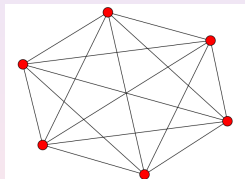
► Check if the graph holds a *Eulerian path*: `is_eulerian()`



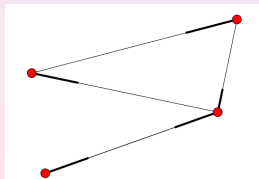
```
nx.is_eulerian(g)
False
```



```
g5 = nx.complete_graph(5)
nx.draw(g5)
nx.is_eulerian(g5)
True
```



```
g6 = nx.complete_graph(6)
nx.draw(g6)
nx.is_eulerian(g6)
False
```

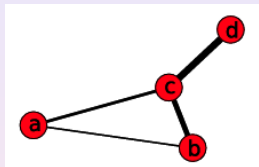


```
h=nx.DiGraph([(0:[3], 1:[2], 2:[3], 3:[0, 1])])
nx.draw(h)
nx.is_eulerian(h)
True
```

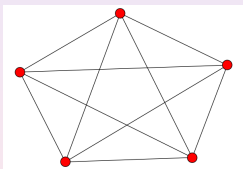
Eulerian path/circuit with NetworkX

How to solve this problem with `python-NetworkX`?

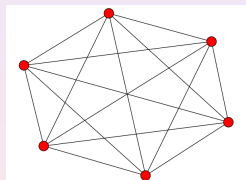
► Check if the graph holds a *Eulerian path*: `is_eulerian()`



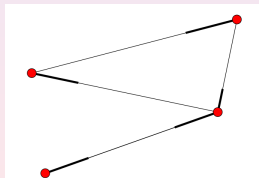
```
nx.is_eulerian(g)
False
```



```
g5 = nx.complete_graph(5)
nx.draw(g5)
nx.is_eulerian(g5)
True
```



```
g6 = nx.complete_graph(6)
nx.draw(g6)
nx.is_eulerian(g6)
False
```

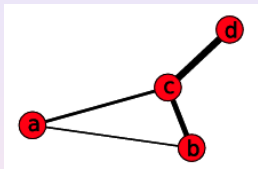


```
h=nx.DiGraph([(0:[3], 1:[2], 2:[3], 3:[0, 1])])
nx.draw(h)
nx.is_eulerian(h)
True
```

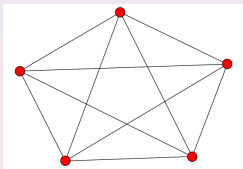
Eulerian path/circuit with NetworkX

How to solve this problem with `python-NetworkX`?

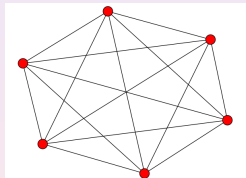
► Check if the graph holds a *Eulerian path*: `is_eulerian()`



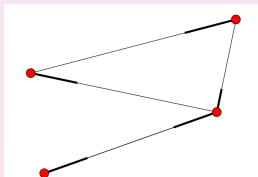
```
nx.is_eulerian(g)
False
```



```
g5 = nx.complete_graph(5)
nx.draw(g5)
nx.is_eulerian(g5)
True
```



```
g6 = nx.complete_graph(6)
nx.draw(g6)
nx.is_eulerian(g6)
False
```

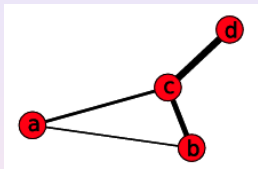


```
h=nx.DiGraph([(0:[3], 1:[2], 2:[3], 3:[0, 1])])
nx.draw(h)
nx.is_eulerian(h)
True
```

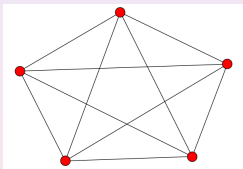
Eulerian path/circuit with NetworkX

How to solve this problem with `python-NetworkX`?

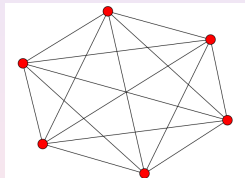
► Check if the graph holds a *Eulerian path*: `is_eulerian()`



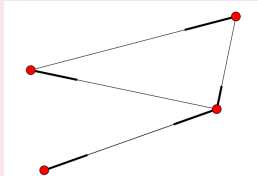
```
nx.is_eulerian(g)
False
```



```
g5 = nx.complete_graph(5)
nx.draw(g5)
nx.is_eulerian(g5)
True
```



```
g6 = nx.complete_graph(6)
nx.draw(g6)
nx.is_eulerian(g6)
False
```



```
h=nx.DiGraph([(0:[3], 1:[2], 2:[3], 3:[0, 1])])
nx.draw(h)
nx.is_eulerian(h)
True
```

Network Topology

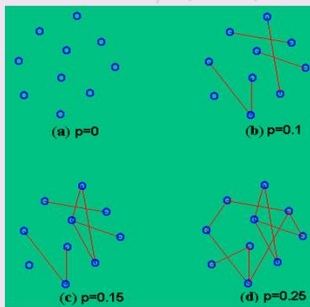
how nodes are *connected*

▶ According to the **topology** is possible to “classify” the networks.

Distribution of links $P(k)$ (degree distrib), avg. nbr. of links $\langle k \rangle$, clustering coeff, avg. length $\langle \ell \rangle$ (diam),...

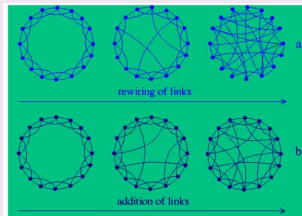
▼ random networks

▶ Erdős-Rényi (1960)



▼ regular or uniform

▶ globally coupling,
nearest-neighbors, ...



▼ irregular or heterog.

▶ non-uniform distrib.
(star), ...



Network Topology

how nodes are *connected*

▶ According to the *topology* is possible to “classify” the networks.

Distribution of links $P(k)$ (degree distrib), avg. nbr. of links $\langle k \rangle$, *clustering coeff*, *avg. length* $\langle \ell \rangle$ (diam),...

▼ random networks

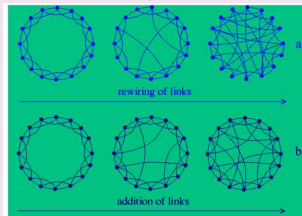
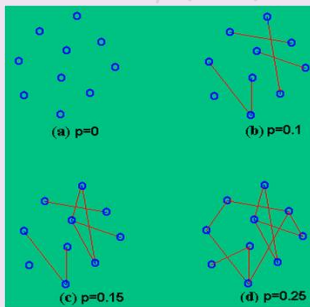
▶ Erdős-Rényi (1960)

▼ regular or uniform

▶ globally coupling,
nearest-neighbors, ...

▼ irregular or heterog.

▶ non-uniform distrib.
(star), ...



Network Topology

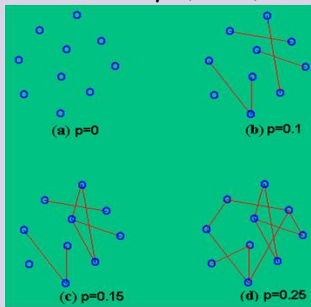
how nodes are *connected*

▶ According to the *topology* is possible to “classify” the networks.

Distribution of links $P(k)$ (degree distrib), avg. nbr. of links $\langle k \rangle$, *clustering coeff*, *avg. length* $\langle \ell \rangle$ (diam),...

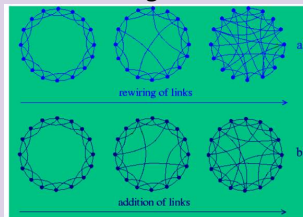
▼ *random networks*

▶ Erdős-Rényi (1960)



▼ *regular or uniform*

▶ globally coupling,
nearest-neighbors, ...



▼ *irregular or heterog.*

▶ non-uniform distrib.
(star), ...



e.g. airlines routes ▶

Complex Networks I

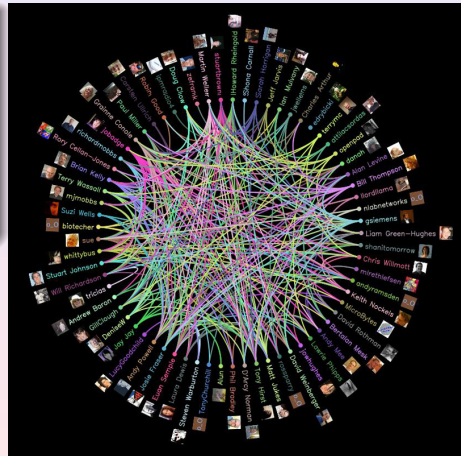
Properties of «REAL» networks

▶ Small-World [Watts and Strogatz, Nature, 1998]

most nodes can be reached by a small number of steps («6 dof»!)

▶ **Connectivity distribution:** uniform but decays exponentially

▶ **Homogeneous nature:** each node has roughly the same number of links



Complex Networks II

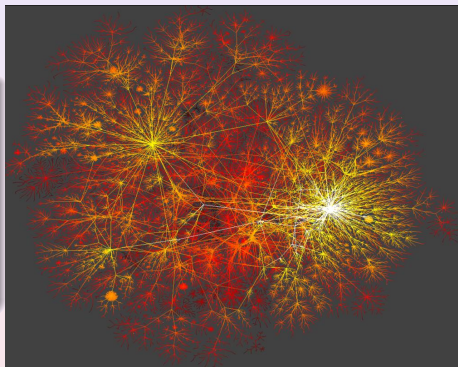
Properties of «REAL» networks

► Scale-Free [Barabási and Albert, Science, 1999]

some nodes act as “highly connected hubs”, and most nodes are of low degree distribution:

$$P(k) = k^{-\gamma}$$

$$2 \leq \gamma \leq 3 \text{ (power law)}$$



MultiGraph: multiple edges

```
mg = nx.MultiGraph()
mg.add_weighted_edges_from([(1,2, .5), (1,2, .75), (2,3, .5)])
mg.degree()
{1: 2, 2: 3, 3: 1}
mg.degree(weight='weight')
{1: 1.25, 2: 1.75, 3: 0.5}
```

DiGraph: directed graphs

```
dg = nx.DiGraph()
dg.add_weighted_edges_from([(1,4,0.5), (3,1,0.75)])
dg.degree(1,weight='weight')
1.25
dg.successors(1)
[4]
dg.predecessors(1)
[3]
```

More graphs generators

```
K_5=nx.complete_graph(5)
nx.draw(K_5)

K_3_5=nx.complete_bipartite_graph(3,5)
nx.draw(K_3_5)

barbell=nx.barbell_graph(10,10)
nx.draw(barbell)

lollipop=nx.lollipop_graph(10,20)
nx.draw(lollipop)
```

Random graphs generators

```
er=nx.erdos_renyi_graph(100,0.15)
nx.draw(er)

ws=nx.watts_strogatz_graph(30,3,0.1)
nx.draw(ws)

ba=nx.barabasi_albert_graph(100,5)
nx.draw(ba)

red=nx.random_lobster(100,0.9,0.9)
nx.draw(red)
```

MultiGraph: multiple edges

```
mg = nx.MultiGraph()
mg.add_weighted_edges_from([(1,2, .5), (1,2, .75), (2,3, .5)])
mg.degree()
{1: 2, 2: 3, 3: 1}
mg.degree(weight='weight')
{1: 1.25, 2: 1.75, 3: 0.5}
```

DiGraph: directed graphs

```
dg = nx.DiGraph()
dg.add_weighted_edges_from([(1,4,0.5), (3,1,0.75)])
dg.degree(1,weight='weight')
1.25
dg.successors(1)
[4]
dg.predecessors(1)
[3]
```

More graphs generators

```
K_5=nx.complete_graph(5)
nx.draw(K_5)

K_3_5=nx.complete_bipartite_graph(3,5)
nx.draw(K_3_5)

barbell=nx.barbell_graph(10,10)
nx.draw(barbell)

lollipop=nx.lollipop_graph(10,20)
nx.draw(lollipop)
```

Random graphs generators

```
er=nx.erdos_renyi_graph(100,0.15)
nx.draw(er)

ws=nx.watts_strogatz_graph(30,3,0.1)
nx.draw(ws)

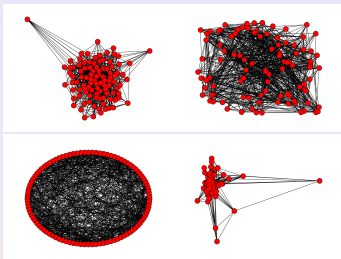
ba=nx.barabasi_albert_graph(100,5)
nx.draw(ba)

red=nx.random_lobster(100,0.9,0.9)
nx.draw(red)
```

Visualization & Graph properties...

Visualization

```
import networkx as nx
import pylab as plt
g = nx.erdos_renyi_graph(100,0.15)
nx.draw(g)
nx.draw_random(g)
nx.draw_circular(g)
nx.draw_spectral(g)
plt.savefig('graph.png')
```



Graph properties

```
N,K = g.order(), g.size()
avg_deg = float(K)/N
print "Nodes:_", N
print "Edges:_", K
print "Average_degree:_", avg_deg
```

Degree distribution

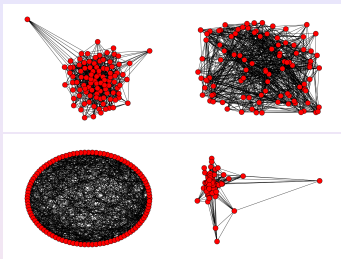
```
g.in_degrees()
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'Graph' object has no attribute 'in_degrees'
```

Visualization & Graph properties...

Visualization

```
import networkx as nx
import pylab as plt
g = nx.erdos_renyi_graph(100,0.15)
nx.draw(g)
nx.draw_random(g)
nx.draw_circular(g)
nx.draw_spectral(g)
plt.savefig('graph.png')
```



Graph properties

```
N,K = g.order(), g.size()
avg_deg = float(K)/N
print "Nodes:_", N
print "Edges:_", K
print "Average_degree:_", avg_deg
```

Degree distribution

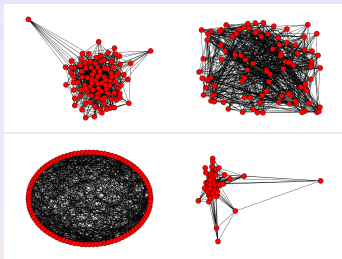
```
g.in_degrees()
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'Graph' object has no attribute 'in_degrees'
```

Visualization & Graph properties...

Visualization

```
import networkx as nx
import pylab as plt
g = nx.erdos_renyi_graph(100,0.15)
nx.draw(g)
nx.draw_random(g)
nx.draw_circular(g)
nx.draw_spectral(g)
plt.savefig('graph.png')
```



Graph properties

```
N,K = g.order(), g.size()
avg_deg = float(K)/N
print "Nodes:_", N
print "Edges:_", K
print "Average_degree:_", avg_deg
```

Degree distribution

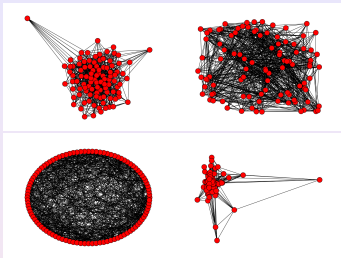
```
g.in_degrees()
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'Graph' object has no attribute 'in_degrees'
```

Visualization & Graph properties...

Visualization

```
import networkx as nx
import pylab as plt
g = nx.erdos_renyi_graph(100,0.15)
nx.draw(g)
nx.draw_random(g)
nx.draw_circular(g)
nx.draw_spectral(g)
plt.savefig('graph.png')
```



Graph properties

```
N,K = g.order(), g.size()
avg_deg = float(K)/N
print "Nodes:_", N
print "Edges:_", K
print "Average_degree:_", avg_deg
```

Degree distribution

```
g.in_degrees()
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'Graph' object has no attribute 'in_degrees'
```

Graph properties: degree distribution ...

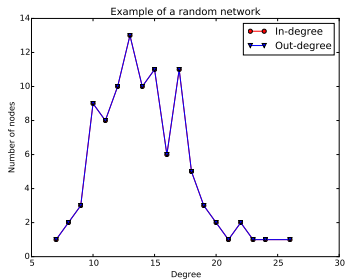
Degree distribution

```

diG=nx.DiGraph(g) #create a directed graph
# account for in/out-going connections...
in_degrees = diG.in_degree() # dictionary node:degree
out_degrees = diG.out_degree() # dictionary node:degree
in_values = sorted(set(in_degrees.values()))
out_values = sorted(set(out_degrees.values()))
# put data in histogram-form
in_hist = [in_degrees.values().count(x) for x in in_values]
out_hist = [out_degrees.values().count(x) for x in out_values]

# plot
plt.figure()
plt.plot(in_values,in_hist,'ro-') # in-degree
plt.plot(out_values,out_hist,'bv-') # out-degree
plt.legend(['In-degree', 'Out-degree'])
plt.xlabel('Degree')
plt.ylabel('Number_of_nodes')
plt.title('Example_of_a_random_network')
plt.savefig('network_degree_distribution.pdf')
plt.close()

```

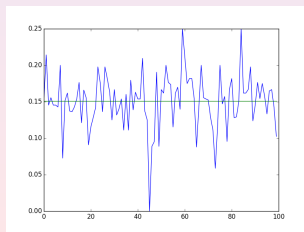
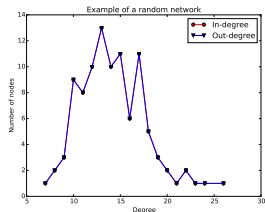
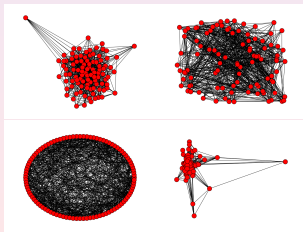


Graph properties: connectivity properties ...

Clustering Coefficient

```
import numpy as np
g_ud = diG.to_undirected()
# Clustering coefficient of node 0
print nx.clustering(g_ud, 0)
# Clustering coefficient of all nodes (in a dictionary)
clust_coefficients = nx.clustering(g_ud)
# Average clustering coefficient
ccs = nx.clustering(g_ud)
avg_clust = sum(ccs.values()) / len(ccs)

plt.plot(ccs.keys(), ccs.values())
plt.plot(ccs.keys(), np.ones(len(ccs)) * avg_clust)
```



Modeling Twitter Relationships I

"Following"

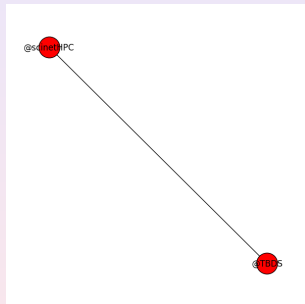
```
import networkx as nx
import pylab as plt

g = nx.Graph() #nx.DiGraph()
g.add_node(0, {"handle": "@scinetHPC"})
g.add_node(1, {"handle": "@TBDS"})
g.add_edge(0, 1)

node_labels = nx.get_node_attributes(g, 'handle')

plt.figure(figsize=(6, 6))
plt.ion()

pos = nx.spring_layout(g)
nx.draw(g, pos, arrows=True, node_size=900)
nx.draw_networkx_labels(g, pos, labels = node_labels)
```



Modeling Twitter Relationships II

"Re-Tweets"

```
g = nx.DiGraph()
g.add_node(0, {"id": "@scinethPC", "color": "r"})
g.add_node(1, {"id": "@TBDS", "color": "r"})
g.add_node(2, {"id": "Tweet_#530038648860065793", "color": "orange"})
g.add_edge(0, 1, {"type": "follows"})
g.add_edge(0, 2, {"type": "retweeted"})

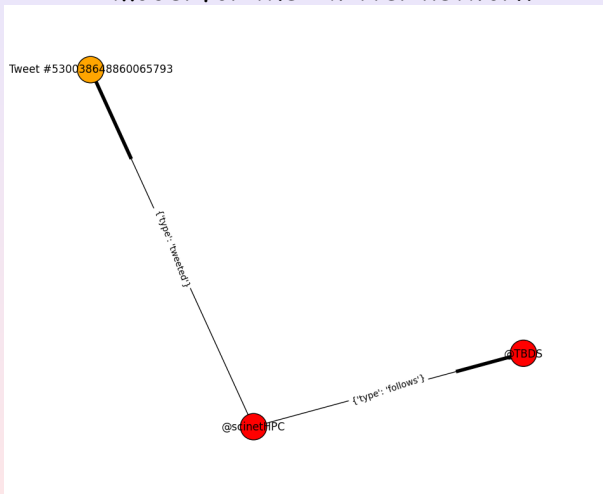
node_labels = nx.get_node_attributes(g, 'id')
edge_labels = nx.get_edge_attributes(g, 'type')
colors = nx.get_node_attributes(g, 'color')

plt.figure(figsize=(6,6))

pos = nx.spring_layout(g)
nx.draw(g, pos, arrows=True, node_size=900, node_color=colors.values())
nx.draw_networkx_labels(g, pos, labels = node_labels)
nx.draw_networkx_edge_labels(g, pos, labels = edge_labels)
```

Modeling Twitter Relationships III

A model for the Twitter network



Modeling the I/O-ops @ SciNet I

Data

tcs-f06n01	tcs-f06n01	0.013554078
tcs-f06n01	tcs-f06n01	0.004655432
tcs-f07n02	gpc-f119n017	0.002597279
tcs-f07n02	tcs-f07n02	0.000283613
tcs-f06n02	tcs-f06n02	0.000703586
tcs-f06n02	tcs-f06n02	0.001286946
tcs-f07n01	tcs-f07n01	0.025607990
tcs-f07n01	tcs-f07n01	0.018086633
tcs-f07n01	tcs-f07n01	0.003623529
tcs-f06n04	tcs-f06n04	0.012281074
tcs-f06n04	tcs-f06n04	0.012600201
tcs-f06n03	tcs-f06n03	0.006405814
tcs-f06n03	tcs-f06n03	0.024002783
tcs-f06n03	tcs-f06n03	0.019571064
tcs-f07n04	tcs-f07n04	0.034180323
tcs-f07n04	tcs-f07n04	0.100195957

449 records

▼ networkX.read_edgelist(...)

g =

```
nx.read_format("path/to/file.txt", ...options..
```

```
nx.write_format(g, "path/to/file.txt", ...options..
```

► Read and write edge lists

```
g = nx.read_edgelist(path, comments='#',
                    create_using=None,
                    delimiter=' ', nodetype=None,
                    data=True, edgetype=None,
                    encoding='utf-8')
```

```
nx.write_edgelist(g, path, comments='#',
                  delimiter=' ', data=True,
                  encoding='utf-8')
```

Modeling the I/O-ops @ SciNet II

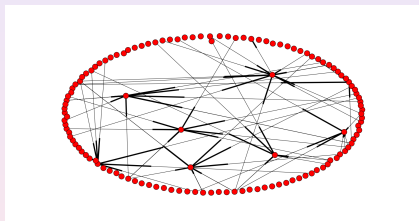
Importing and plotting the data

```
import networkx as nx
import pylab as plt

scinetG = nx.read_edgelist("scinet_data.txt",
    comments='#',
    create_using=nx.MultiDiGraph(),
    delimiter='_',
    nodetype=str, data=(('weight', float),) )

plt.ion()
nx.draw(scinetG)

plt.figure()
```



Modeling the I/O-ops @ SciNet III

"Massaging" the network (data)

```
plt.figure()
plt.axis('off')

cutOff=0.001
elarge=[(u,v) for (u,v,d) in scinetG.edges(data=True) if d['weight'] > cutOff]
esmall=[(u,v) for (u,v,d) in scinetG.edges(data=True) if d['weight'] <= cutOff]

#try different ones from the ones below...
#pos=nx.spring_layout(scinetG) # positions for all nodes
pos=nx.graphviz_layout(scinetG)
#pos=nx.shell_layout(scinetG)
#pos=nx.random_layout(scinetG)
#pos=nx.spectral_layout(scinetG)

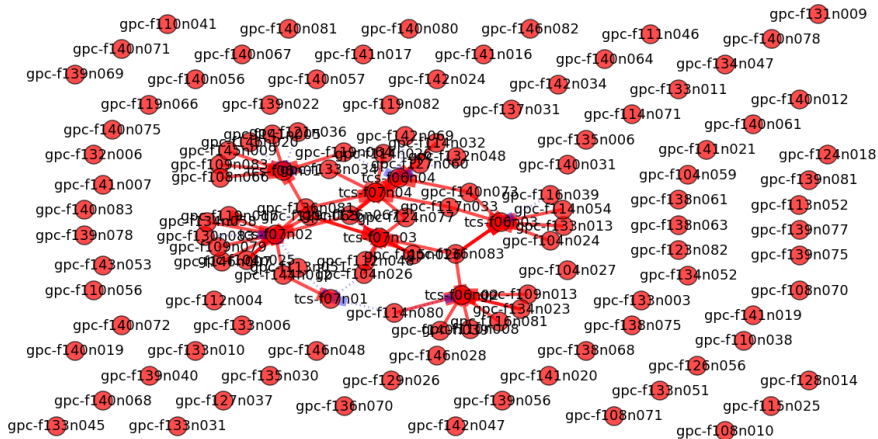
# nodes
nx.draw_networkx_nodes(scinetG, pos,
#                       node_color=nodeType.values(),
#                       node_size=tmW.values(),
                       cmap=plt.cm.Reds, alpha=0.7)

# edges
color_edges=range(scinetG.number_of_edges())
nx.draw_networkx_edges(scinetG, pos, edgelist=elarge,
#                       width=3, alpha=0.7, edge_color='r',
#                       with_labels='True', edge_cmap=plt.cm.Blues)
nx.draw_networkx_edges(scinetG, pos, edgelist=esmall,
#                       width=2, alpha=0.4, edge_color='b', style='dotted')
```

Modeling the I/O-ops @ SciNet IV

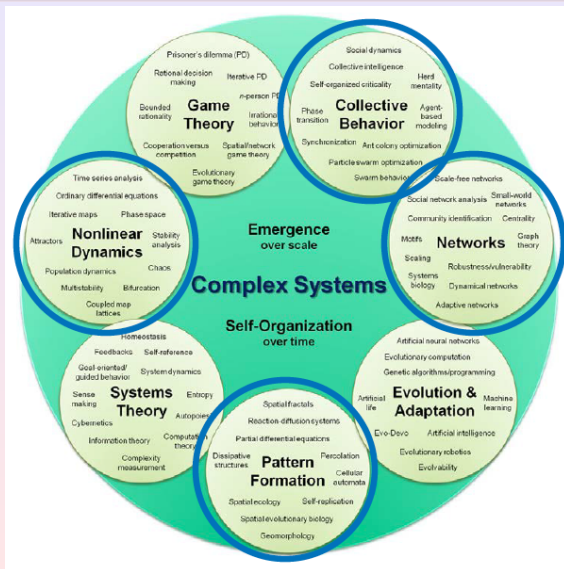
```
# labels
try:
    nx.draw_networkx_labels(scinetG, pos, labels, font_size=13,
                           font_family='sans-serif', font_color='Indigo')
except:
    nx.draw_networkx_labels(scinetG, pos, font_size=13, font_family='sans-serif')
```


Modeling the I/O-ops @ SciNet V



PyCX: Python-based Complex systems simulations

Online repository: <http://pycx.sf.net/>



Further Resources & Examples

- ▼ NetworkX: Network Analysis with Python, Salvatore Scellato
<http://www.cl.cam.ac.uk/~cm542/teaching/2011/stna-pdfs/stna-lecture11.pdf>
- ▼ Graph Theory, including modelling of relationships, tweets, how to build a NetworkX graph from Twitter crawl results ...
http://nbviewer.ipython.org/github/jquacinella/IS609_Group/blob/master/GraphTheory/Graph%20Theory%20-%20Social%20Networking.ipynb
- ▶ Graph and Network Analysis
<http://mlg.ucd.ie/files/summer/tutorial.pdf>
- ▶ Social Network Analysis in Python
<https://ep2013.europython.eu/media/conference/slides/social-network-analysis-in-python.pdf>
- ▶ Tutorial on PyCX
<http://pycx.sourceforge.net/PyCX-ECAL2013-tutorial.pdf>

More about Complex Networks

NECSI:

<http://necsi.edu/> <http://www.necsi.edu/publications/dcs/index.html>

<http://www.necsi.edu/events/vidlib/> <http://www.necsi.edu/research/overview.php>

Some other good references:

<http://www-personal.umich.edu/~mejn/courses/2004/cscs535/review.pdf>

http://www.ifr.ac.uk/netsci08/Download/Invited/ws1_Caldarelli.pdf

<http://barabasilab.neu.edu/courses/phys5116/>

SantaFe Institute:

<http://www.santafe.edu/education/schools/complex-systems-summer-schools/2015-program-info/>

<http://www.santafe.edu/education/schools/complex-systems-summer-schools/2015-program-info/>

