

# **Graph Mining and Analysis with Python**

Master Data Science - MVA

Fragkiskos Malliaros, Antoine Tixier and Michalis Vazirgiannis

Thursday, 12 February 2015

# Graph Mining with NetworkX

---

- **NetworkX** is a Python package for creating and manipulating graphs and networks

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

- Main features

- Python language data structures for graphs, digraphs, and multigraphs
- Nodes can be **anything** (e.g., text, images, XML records)
- Edges can hold arbitrary data (e.g. weights, time-series)
- Generators for classic graphs, random graphs, and synthetic networks
- Standard graph algorithms
- Network structure and analysis measures
- Open source [BSD license](#)
- Well tested: more than 1800 unit tests, >90% code coverage
- Additional benefits from Python: fast prototyping, easy to teach, multi-platform

# Creating Undirected Graphs

---

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

Import library

```
>>> G = nx.Graph()
```

Create a new undirected graph

```
>>> G.add_node("Jim")
>>> G.add_node("Jenny")
>>> G.add_node("David")
>>> G.add_edge("Jim", "David")
>>> G.add_edge("Jenny", "David")
```

Add new nodes and edges

```
>>> print G.number_of_nodes()
>>> print G.number_of_edges()
>>> print G.nodes()
>>> print G.edges()
```

Print the basic characteristics of the graph

```
>>> print G.degree("Jim")
>>> print G.degree()
```

Compute the degree of a specific node or of all the nodes in the graph

# Creating Directed Graphs

---

```
>>> G = nx.DiGraph()
```

Create a new directed graph

```
>>> G.add_edges_from([("A","B"), ("C","A")])
```

Add nodes and edges

```
>>> print G.in_degree()
>>> print G.out_degree()
```

Print the in-degree and out-degree of the nodes

```
>>> print G.neighbors("A")
>>> print G.neighbors("B")
```

Print the neighborhood nodes of A and B

```
>>> U = G.to_undirected()
>>> print U.neighbors("B")
```

Convert the directed graph to undirected and print the neighbors of B

# Loading/Writing Graphs from/to File

**test.txt**

```
a b
b c
b d
c d
```

Suppose that you have a graph stored in a text file in the **edge list** format: node pairs, one edge per line

```
>>> G = nx.read_edge_list("test.txt")
```

----

**Syntax:**

```
>>> G = nx.read_adjlist("adjlist.txt", comments='#',
delimiter=' ', nodetype=int)
```

**Especially for character type nodes:**

```
G1=nx.read_edgelist("test.txt", comments='#', delimiter=' ',
nodetype=None, create_using=nx.Graph())
```

Add a new edge between nodes **a** and **c** of **G** and save the new graph into the “edgelist.txt” file

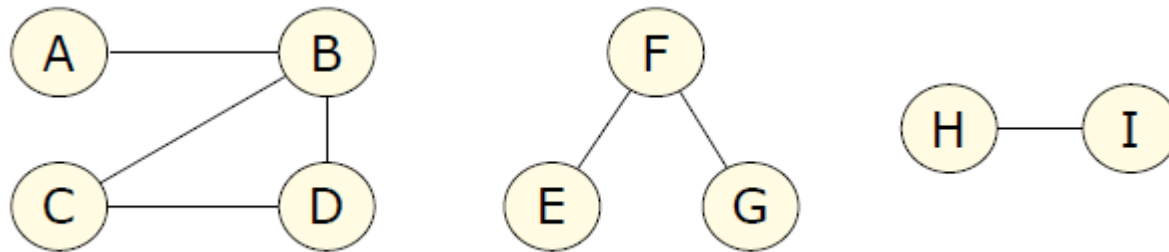
```
>>> G.add_edge(a,c)
```

```
>>> nx.write_edgelist (G, "edgelist.txt", delimiter=' ')
```

# Graph Connectivity

---

- A graph is connected if there is a path between every pair of nodes in the graph
- A connected component is a subset of the nodes where
  - A path exists between every pair in the subset



**Example graph with 3 connected components**

# Connectivity in NetworkX

## Define the graph

```
G = nx.Graph()
G.add_edges_from([("a", "b"), ("b", "c"), ("b", "d"), ("c", "d")])
G.add_edges_from([("e", "f"), ("f", "g"), ("h", "i")])
```

## Examine if the graph is connected and if not, find the number of con. components

```
>>> print nx.is_connected(G)
False
>>> print nx.number_connected_components(G)
3
```

## Find all connected components and print their nodes

```
>>> comps = nx.connected_component_subgraphs(G)
>>> print comps[0].nodes()
['a', 'c', 'b', 'd']
>>> print comps[0].nodes()
['e', 'g', 'f']
>>> print comps[2].nodes()
['i', 'h']
```

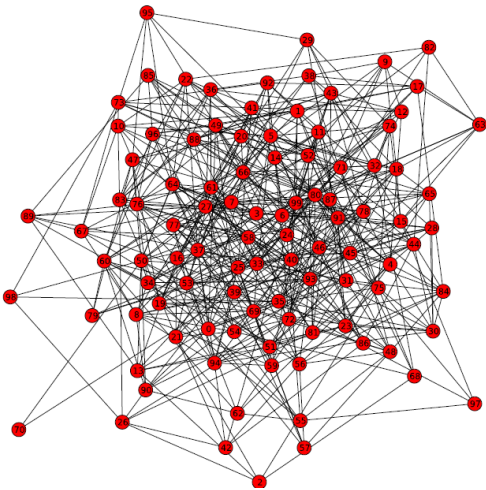
# Network Visualization

- NetworkX uses the **Matplotlib** module for some simple network visualizations

**Create a random graph based on the Erdos-Renyi model and visualize it**

```
import networkx as nx
import matplotlib.pyplot as plt

G = nx.erdos_renyi_graph(100, 0.11)
plt.figure(figsize=(10, 10))
nx.draw(G)
plt.show()
```



More examples of graph visualization at:  
<http://networkx.github.io/documentation/latest/gallery.html>