CMSC5733 Social Computing

Tutorial 1: NetworkX & Graphviz

Shenglin Zhao

The Chinese University of Hong Kong

slzhao@cse.cuhk.edu.hk

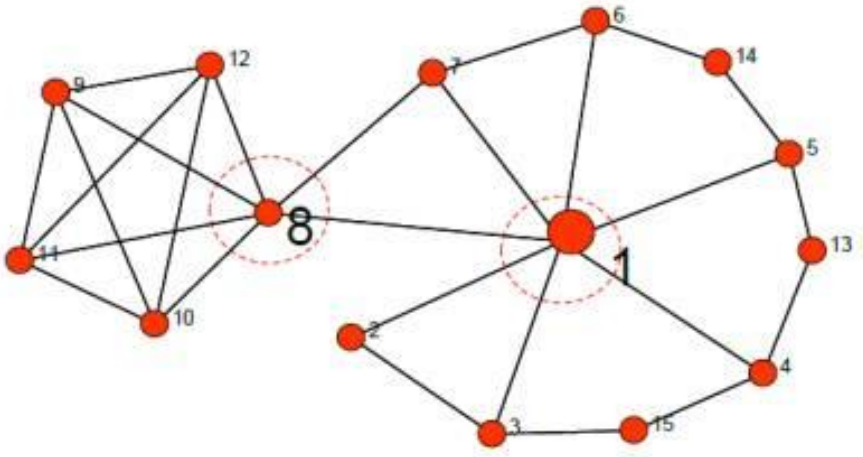
Tutorial Overview

- NetworkX
  - Creating a graph
  - Adding attributes
  - Directed Graphs
  - Graph generators
  - Analyzing graphs
  - Drawing graphs
- Graphviz
  - Dot language
  - Sample graphs

NetworkX

What is it?

- A Python language software package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks



NetworkX

What can it do ?

- Load and store networks
- Generate random and classic networks
- Analyze network structure
- Build network models
- Draw networks
- …

Installing NetworkX

- Quick install
  - Python Egg: https://pypi.python.org/pypi/networkx
  - pip install networkx or easy_install networkx
- Current version: 1.10

Creating A Graph

- Create an empty graph with no nodes

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

- In NetworkX, nodes can be any hashable object, e.g., a text string, an image, and xml object, another graph, a customized node object, etc.
- Show the graph

```
>>> import matplotlib.pyplot as plt
>>> nx.draw(G)
>>> plt.show()
```

Nodes

- Add one node at a time

```
>>> G.add_node(1)
```

- Add a list of nodes

```
>>> G.add_nodes_from([2,3])
```

Nodes

- Add nbunch of nodes
  - An nbunch is any iterable container of nodes that is not itself a node in the graph
  - E.g., a list, set, graph, file, etc.

```
>>> H=nx.path_graph(10)
>>> G.add_nodes_from(H)
```

Add a graph as a node

```
>>> G.add_node(H)
```

Edges

- Add one edge at a time

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

- Add a list of edges

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

- Add any ebunch of edges.

```
>>> G.add_edges_from(H.edges())
```

Demolishing A Graph

- Functions
  - remove_node()
  - remove_nodes_from()
  - remove_edge()
  - remove_edges_from()
  - clear()

Graph Properties

- Functions
  - nodes()
  - number_of_nodes()
  - edges()
  - number_of_edges()
  - neighbors()
  - degree()
  - …

Adding Attributes

- Attributes
  - Weights, labels, colors, etc.

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

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

Node and Edge Attributes

```
>>> G.add_node(1, time='5pm')
>>> G.add_nodes_from([3], time='2pm')
>>> G.node[1]
{'time': '5pm'}
>>> G.node[1]['room'] = 714
```

```
>>> G.add_edge(1, 2, weight=4.7 )
>>> G.add_edges_from([(3,4),(4,5)], color='red')
>>> G.add_edges_from([(1,2,{'color':'blue'}), (2,3,{'weight':8})])
>>> G[1][2]['weight'] = 4.7
>>> G.edge[1][2]['weight'] = 4
```

Directed Graph

- Additional functions
  - out_edges(), in_degree()
  - predecessors(), successors()

```
>>> DG=nx.DiGraph()
>>> DG.add_weighted_edges_from([(1,2,0.5), (3,1,0.75)])
>>> DG.out_degree(1,weight='weight')
0.5
>>> DG.degree(1,weight='weight')
1.25
>>> DG.successors(1)
[2]
>>> DG.neighbors(1)
[2]
```

Graph Generators

- Applying classic graph operations
  - subgraph(G, nbunch), induce subgraph 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
  - create_empty_copy(G), return an empty copy of the same graph class
  - complement(G)-graph complement
  - convert_to_undirected(G), return an undirected representation of G
  - convert_to_directed(G), return a directed representation of G

Graph Generators (cont.)

- Calling graph generators

```
>>> petersen=nx.petersen_graph()
>>> tutte=nx.tutte_graph()
>>> maze=nx.sedgewick_maze_graph()
>>> tet=nx.tetrahedral_graph()
```

```
>>> 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)
```

```
>>> 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)
```

Refer to http://networkx.lanl.gov/reference/generators.html for details and more examples

Graph Generators

- Loading from files
  - common graph formats: edge lists, adjacency lists, GML, GraphML, pickle, LEDA, etc.

```
>>> G=nx.path_graph(4)
>>> nx.write_adjlist(G, "test.adjlist")
>>> G=nx.read_adjlist("test.adjlist")
```

Analyzing Graphs

- Example, compute degree

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

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

```
>>> nx.degree(G)
{1: 2, 2: 1, 3: 1, 'spam': 0}
```

- More examples, computing the betweenness, closeness, community, etc., see reference http://networkx.lanl.gov/reference/algorithms.html

Drawing Graphs - Matplotlib

- Install Matplotlib, pip or easy_install
- Import Matplotlib's plot interface

```
>>> import matplotlib.pyplot as plt
```

- Draw

```
>>> nx.draw(G)
```

- Show the graph

```
>>> plt.show()
```

- Save

```
>>> plt.savefig("path.png")
```

Graphviz and PyGraphviz

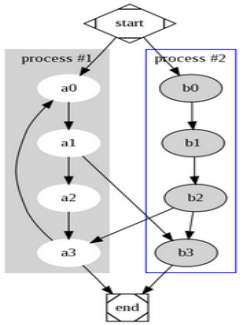
- An open source graph visualization software for graph drawing and graph layout

Welcome to Graphviz

Available translations: Romanian, Russian, Russian (more natural?), Serbo-Croatian, Bulgarian Home and About

What is Graphviz?

Graphviz is open source graph visualization software. Graph visualization is a way of representing structural information as diagrams of abstract graphs and networks. It has important applications in networking, bioinformatics, software engineering, database and web design, machine learning, and in visual interfaces for other technical domains.



Graphviz

- Install
  - Windows, http://www.graphviz.org/Download_windows.php
  - Mac, mountainlion version, http://www.graphviz.org/Download_macos.php
  - Linux, http://www.graphviz.org/Download..php
- Features
  - Take descriptions of graphs in a simple text language
  - Make diagrams in useful formats
  - Colors, fonts, tabular node layouts, line styles, hyperlinks, etc.

The Dot Language

graph : [**strict**] (**graph** | **digraph**) [*ID*] '**{**' *stmt_list* '**}**'
stmt_list : [*stmt* ['**;**'] [*stmt_list*]]
stmt : *node_stmt*
| *edge_stmt*
| *attr_stmt*
| *ID* '**=**' *ID*
| *subgraph*
attr_stmt : (**graph** | **node** | **edge**) *attr_list*
attr_list : '**[**' [*a_list*] '**]**' [*attr_list*]
a_list : *ID* ['**=**' *ID*] ['**,**'] [*a_list*]
edge_stmt : (*node_id* | *subgraph*) *edgeRHS* [*attr_list*]
edgeRHS : *edgeop* (*node_id* | *subgraph*) [*edgeRHS*]
node_stmt : *node_id* [*attr_list*]
node_id : *ID* [*port*]
port : '**:**' *ID* ['**:**' *compass_pt*]
| '**:**' *compass_pt*
subgraph : [**subgraph** [*ID*]] '**{**' *stmt_list* '**}**'
compass_pt : (**n** | **ne** | **e** | **se** | **s** | **sw** | **w** | **nw** | **c** | **_**)

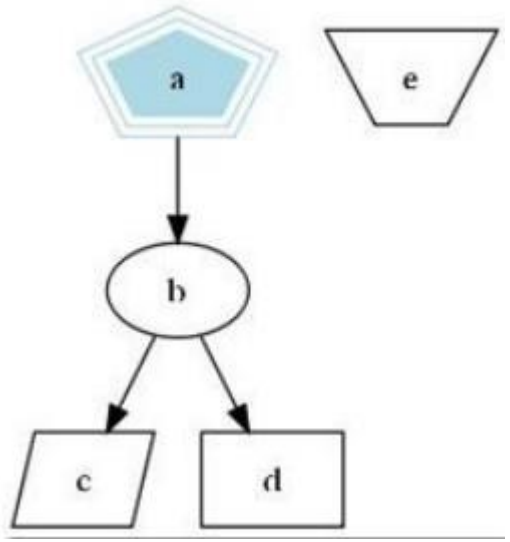
- Refer to http://www.graphviz.org/content/dot-language for more details

Sample Graphs

```
digraph G{
        size = "4, 4"
        a->b->c;
        b->d;

        a[shape = polygon, sides = 5, peripheries=3,
        color = lightblue, style = filled];
        c[shape = polygon, sides = 4, skew= 0.4,
        lable = "hello world"];
        d[shape = invtriange];
        e[shape = polygon, side = 4, distortion = .7];

}
```



command: dot [file_name] –Tpng –o s.png –Gsplines=line

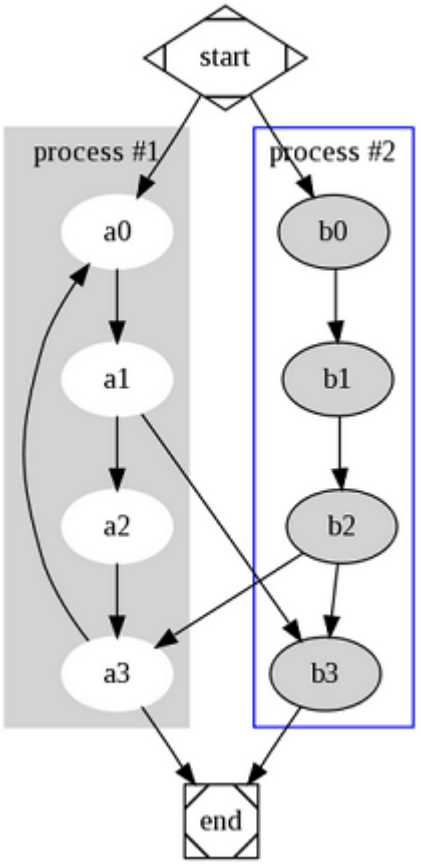
Sample Graphs

```
digraph G {

        subgraph cluster_0 {
                style=filled;
                color=lightgrey;
                node [style=filled,color=white];
                a0 -> a1 -> a2 -> a3;
                label = "process #1";
        }

        subgraph cluster_1 {
                node [style=filled];
                b0 -> b1 -> b2 -> b3;
                label = "process #2";
                color=blue
        }
        start -> a0;
        start -> b0;
        a1 -> b3;
        b2 -> a3;
        a3 -> a0;
        a3 -> end;
        b3 -> end;

        start [shape=Mdiamond];
        end [shape=Msquare];
}
```



command: dot [file_name] –Tpng –o s.png

PyGraphviz

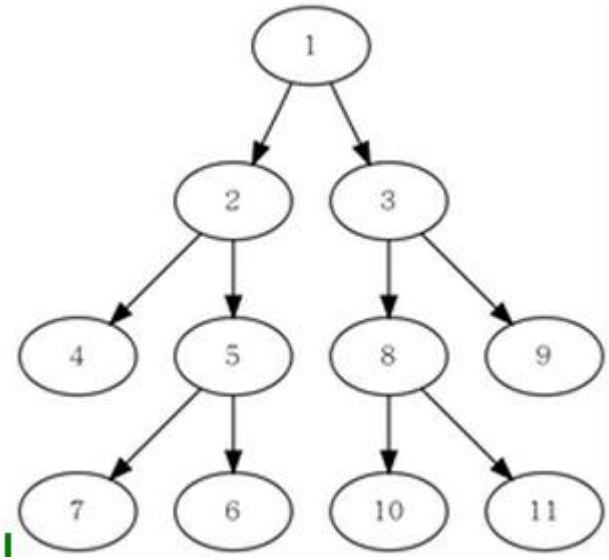
- PyGraphviz is a Python interface to the Graphviz graph layout and visualization package. With PyGraphviz you can create, edit, read, write, and draw graphs using Python to access the Graphviz graph data structure and layout algorithms. PyGraphviz provides a similar programming interface to NetworkX.

- Install, pip or easy_install

Example

```
import pygraphviz as pgv

A=pgv.AGraph(directed=True,strict=True)
A.add_edge(1,2)
A.add_edge(1,3)
A.add_edge(2,4)
A.add_edge(2,5)
A.add_edge(5,6)
A.add_edge(5,7)
A.add_edge(3,8)
A.add_edge(3,9)
A.add_edge(8,10)
A.add_edge(8,11)
A.graph_attr['epsilon']='0.001'
print A.string() # print dot file to standard output
A.write('fooOld.dot')
A.layout('dot') # layout with dot
A.draw('fooOld.png') # write to file
```



References

- http://networkx.github.io/documentation/latest/index.html
- http://networkx.github.io/documentation/latest/tutorial/index.html
- http://www.graphviz.org/
- https://pypi.python.org/pypi/pygraphviz/
- http://www.cnblogs.com/rocketfan/archive/2009/09/09/1563628.html
- http://www.cnblogs.com/sld666666/archive/2010/06/25/1765510.html