The goal of this lab is to understand how to represent structures, dynamics and functions of complex networks in Python using the NetworkX module. NetworkX provides data structures to represent many types of networks and graphs. Graph nodes (up to 100M) are represented as any python objects and edges can contain any arbitrary data.

Make sure your machine runs Linux or reboot it.

Exercise 1: Graphs in Python

Connect using ssh to poseidon.it.usyd.edu.au and ask your tutor for the credentials.

```
comp5313@labmachine:~$ ssh poseidon.it.usyd.edu.au
```

Type python in a terminal window to open the python interpreter:

```
comp5313@poseidon:~$ python
```

Python 2.7.3 (default, Feb 27 2014, 19:58:35)

[GCC 4.6.3] on linux2

Type "help", "copyright", "credits" or "license" for more information.

```
>>> 
```

You can now type python code. First, import the NetworkX module and use a simplifying naming convention (e.g., NX) for it before creating an empty graph on the next line.

```
import networkx as NX

g = NX.Graph()
```

Nodes. To add 1 node to the graph g type: `g.add_node(1)` and to add a list of nodes, types:

```
g.add_nodes_from([2,3])
```

You can also create a container of nodes `h = nx.path_graph(10) and add it to the graph using g.add_nodes_from(h). You can finally remove any node of the graph using g.remove_node(2).

Edges. To add a single edge, type `g.add_edge(1, 2)` or you can define an edge type `e=(2, 3)` and unpack it and add it to the graph as follows: `g.add_edge(*e)`. You can also add a list of edges by typing:

```
g.add_edges_from([(1,2),(1,3)])
```

or equivalently:

```
g.add_path([2, 1, 3])
```
and define a container of edge and add it to the graph with `g.add_edges_from(h.edges())` or remove any edge of the graph with `g.remove_edge(1,2)`.

**Iterating.** To iterate over multiple nodes:

```python
g.add_edge(1,2)
for node in g.nodes():
    print node, g.degree(node)
```

One can similarly iterate over the set of edges by using `edges()` instead of `nodes()`.

**Searching.** Both `g.number_of_nodes()` or `g.order()` return the number of nodes, both `g.number_of_edges()` and `g.size()` return the number of edges, and `g.nodes` (resp. `g.edges`) simply returns an array of all nodes (resp. edges) of `g`. One can obtain the neighbour of a specific node of the graph `g` by typing `g.neighbors(1)` and obtain the degree of this same node with `g.degree(1)`.

![Figure 1: The resulting graph](image)

Create the graph presented in Figure 1 and check by writing up to two lines of python that the graph `g` you obtained is the one represented in Figure 1. What functions did you use?

*Duration: 20 min*

### Exercise 2: Graph properties

Compute the average degree of the graph represented in Figure 1.

What is the clustering coefficient of this graph?

Verify that this is correct by comparing your results to the following program:

```python
coefficients = NX.clustering(g)
clustering_coefficient = sum(coefficients.values()) / len(coefficients)
print clustering_coefficient
```

*Duration: 10 min*
Exercise 3: Graph operations

NetworkX features the common operations on graphs as follows:

- subgraph \((G, \text{bunch})\) returns a subgraph of \(G\) with nodes in \(\text{bunch}\).
- union \((G_1, G_2)\) returns the union graph of two graphs.
- disjoint_union \((G_1, G_2)\): returns the union graph assuming that all nodes are distinct.
- cartesian_product \((G_1, G_2)\):
- compose \((G_1, G_2)\): combine graphs identifying nodes common to both
- complement \((G)\): graph complement
- create_empty_copy \((G)\): return an empty copy of the same graph class
- convert.convert_to_undirected \((G)\): return an undirected version of \(G\).
- convert.convert_to_directed \((G)\): return a directed representation of \(G\).

It can also be used to generate random graphs based on different models. For example, one can generate a Watts-Strogatz graph \([1]\) that has small-world properties, including a short average path length and high clustering coefficients, with

\[
\text{ws}=\text{nx.watts_strogatz_graph}(n,k,p[,s])
\]

where \(n\) is the number of nodes, \(k\) is the number of edges and \(p\) is the probability of rewiring each edge and \(s\) is an optional seed for the pseudo-random number generator.

Draw the graph that results from:

```python
1  bunch=[0, 1, 2, 3, 4]
2  g2 = NX.subgraph(g, bunch)
```

Duration: 10 min

Exercise 4: Advanced properties

Take a look at algorithms that NetworkX offers as presented in the online documentation: [http://networkx.github.io/documentation/networkx-1.9.1/reference/algorithms.html](http://networkx.github.io/documentation/networkx-1.9.1/reference/algorithms.html) and lists one algorithm to compute a property defined in class.

Duration: 10 min

References