Published in Python in Plain English

Valentina Alto Follow
Apr 11, 2020 · 6 min read · Member-only

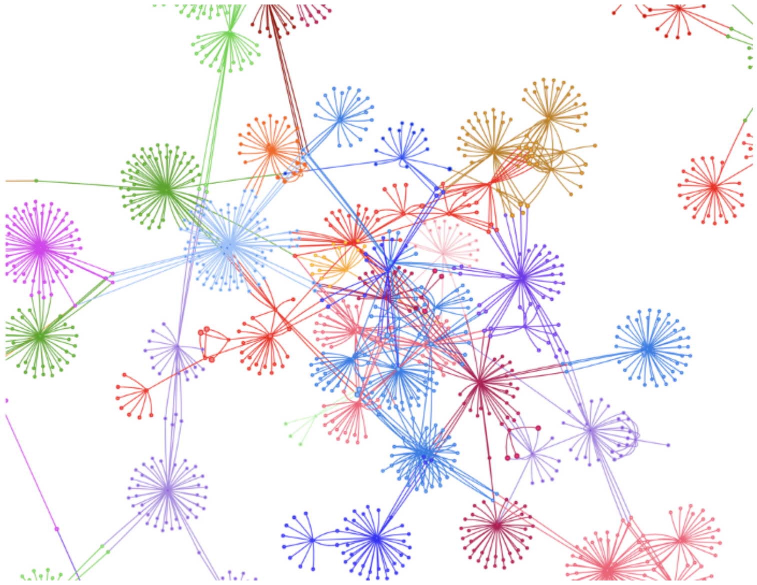


Image source: http://carrefax.com/articles-blog/2018/10/14/case-law-network-graph

Introduction to Network science with NetworkX

Part 2: NetworkX and different types of graphs

In Part 1 of this series, I've been introducing the math behind network science: I've been talking about the concepts of degree centrality, shortest paths, betweenness distribution and so forth.

In this article, I'll rather focus on the Python code to generate and visualize graphs. More specifically, I'm going to dwell on three different types of graphs, which are:

- random graphs
- preferential attachments graphs
- small world graphs

We will see how each category is characterized by specific statistical properties that distinguish them.

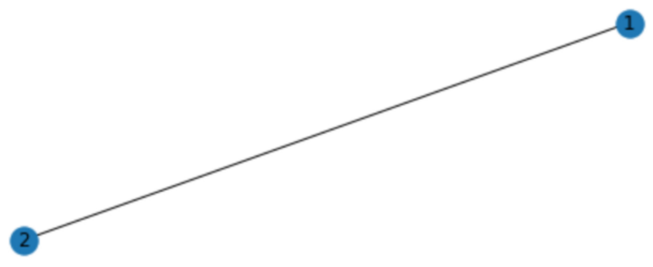
However, before deep diving into those definitions, let's first introduce the Python package we are going to use.

Introduction to NetworkX

According to the official documentation, NetworkX is "a package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks". It can be easily installed via *pip* and then imported into your notebook.

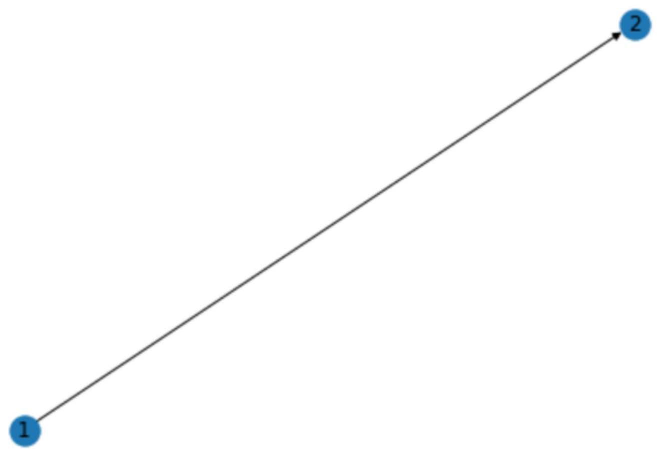
So let's build our first graph step by step:

```
G = nx.Graph()
#adding one node
G.add_node(1)
#adding a second node
G.add_node(2)
#adding an edge between the two nodes (undirected)
G.add_edge(1,2)
nx.draw(G, with_labels=True)
```



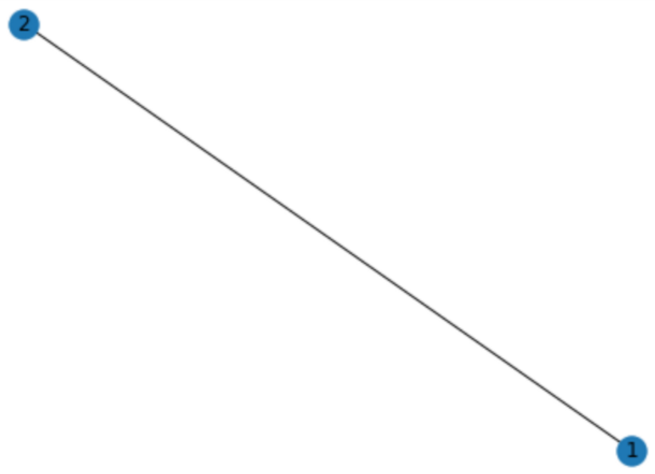
We can also create a directed graph (if you are not familiar with the concept of directed and undirected graphs, check the first part here).

```
DG = nx.DiGraph() #initializing a directed graph
DG.add_node(1)
#adding a second node
DG.add_node(2)
#adding an edge between the two nodes (undirected)
DG.add_edge(1,2)
nx.draw(DG, with_labels=True)
```



Finally, if provided with a directed graph, we can easily convert it to an undirected one (but not the other way around):

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



As you can see, you can create node by node, edge by edge your graph from zero with NetworkX. However, there are more interesting functionalities of this package that allow users to create graphs according to predefined parameters and shapes, and we are going to examine them while talking about the different types of graphs.

Random Graph

As the name suggests, a random graph is a set of nodes connected by links in a purely random way.

```
G_random = nx.gnp_random_graph(n = 30, p = 0.2)
#n=number of nodes, p=probability of edge creation
nx.draw_random(G_random)
```

In this case, we set a probability of edge creation equal to 20%. We can try to lower/increase it to obtain different format of graphs (this time with size=10):

```
p = [0.1, 0.4, 0.6, 0.8]
graphs = [nx.gnp_random_graph(10,i) for i in p]

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(15,15))
```

```
    ax[i].set_axis_off()
    ax[i].set_title('Random graph with probability {}'.format(p[i]))

plt.show()
```

As you can see, when the probability is low the graph looks sparse, while with p=0.8 it is almost complete (consequently, we obtain the complete graph when p=1).

Preferential Attachment graph

This category differs from the latter in the way links are generated. Indeed, in the preferential attachment graph, when new nodes are added, the probability for them to be connected to a well-connected node is greater than that of being connected with a poorly-connected node. In other words, if in a graph there is node A with degree=5 and node B with degree=2, when node C enters the graph, the probability of linking with A is greater than that of linking with B. As a result, the final graph will have few nodes with very high degrees, while many nodes with low degrees.

Let's see how NetworkX can represent those graphs. The method is called *barabasi_albert_graph* after the names of those who approached this work first (Barabasi and Albert, 1999):

```
G_pref = nx.barabasi_albert_graph(n=50, m=2) #where n=number of nodes,
m=number of edge for each additional node
nx.draw_circular(G_pref) #using a ciruclar layout
```

As you can see, in the top-right area there is a denser condensation of links, while the bottom-left nodes exhibit a lower degree.

Let’s inspect the difference between this graph and a random graph (using also, in this case, a circular layout):

```
graphs = [nx.gnp_random_graph(50,0.2), nx.barabasi_albert_graph(n=50,
m=2)]
names = ['Random Graph', 'Preferential Attachment Graph']
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(20,10))
ax = axes.flatten()

for i in range(2):
    nx.draw_circular(graphs[i], ax=ax[i])
    ax[i].set_axis_off()
    ax[i].set_title(names[i])

plt.show()
```

Preferential attachment graphs are a mathematical representation of the situation where “rich get richer and poor get poorer”, where “rich” are those nodes with a high degree.

Note that the preferential attachment mechanism is the one used to generate the so-called *scale-free networks*.

A scale-free network is a network whose degree distribution follows a power law. That is, the fraction $P(k)$ of nodes in the network having k connections to other nodes goes for large values of k as where *alpha* is a parameter whose value is typically in the range (2,3).

The idea of scale-free networks is that, within the system, there are few important nodes that give robustness to the structure, while the majority of nodes are not as indispensable.

Small World Graph

A small world graph defines a structure where many nodes are connected to nearby nodes, while some nodes are connected with nodes that are far away. The name of this mathematical structure reflects the peculiar reality of acquaintances' connections among human beings, whose study traces back to 1967, when Stanley Milgram, a social psychologist at Harvard University, ran a very interesting experiment.

He sent hundreds of letters to unknown people living in Nebraska, asking them to deliver the letter to a stockbroker in Boston. It turned out that the average distance between those people and the stockbroker (totally unknown to them), in terms of the number of people between the first recipient and the final recipient (the broker) was 6 people. In other words, it takes, on average, only “6 people” to reach a completely unknown individual.

So let’s visualize the structure of a small world graph:

```
G_small_world = nx.watts_strogatz_graph(n=100, k = 5, p = 0.4)
#n=number of nodes, k=number of nearby links, p = probability of
rewiring a link to a far away node

graphs = [G_small_world, G_small_world]
names = ['Random view', 'Circular view']
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(30,15))
ax = axes.flatten()

nx.draw_random(graphs[0], ax=ax[0])
ax[0].set_axis_off()
ax[0].set_title(names[0])

nx.draw_circular(graphs[1], ax=ax[1])
ax[1].set_axis_off()
ax[1].set_title(names[1])

plt.show()
```

Conclusion

We have been focusing on three different categories of graphs, each useful in describing different phenomena. Let’s have a final look at the three representations and their degree distributions:

```
import collections

graphs = [nx.gnp_random_graph(50,0.2), nx.barabasi_albert_graph(n=50, 
m=2), nx.watts_strogatz_graph(n=50, k = 5, p = 0.4)]
names = ['Random Graph', 'Preferential Attachment Graph', 'Small World 
Graph']
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(30,15))
ax = axes.flatten()

for i in range(3):
    nx.draw_random(graphs[i], ax=ax[i])
    ax[i].set_axis_off()
    ax[i].set_title(names[i])

for i in range(3, 6):
    nx.draw_circular(graphs[i-3], ax=ax[i])
    ax[i].set_axis_off()

for i in range(6, 9):
    degree_sequence = sorted([d for n, d in graphs[i-6].degree()], 
reverse=True)
    degreeCount = collections.Counter(degree_sequence)
    deg, cnt = zip(*degreeCount.items())
    ax[i].bar(deg, cnt, width=0.80, color='b')

plt.show()
```

As you can see, the degree distribution of the preferential attachment graph is exponential: only few nodes are really important within the system (aka they have many connections with other nodes), while the majority of nodes is purely connected and, if removed, they wouldn’t undermine the whole system.

I hope you enjoyed the reading of the second part of this series. In the next section, I will discuss a real application of networks, by modeling the dynamics of a phenomenon. So stay tuned for Part 3!

References:

- https://it.wikipedia.org/wiki/Stanley_Milgram
- https://networkx.github.io/documentation/networkx-1.10/tutorial/tutorial.html

Read more from Python in Plain English

Recommended from Medium

Shuo Wang

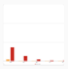
Portfolio Analysis Basics: Returns and Drawdowns

READ/DOWNLOAD$# Intelligent ...

READ/DOWNLOAD$# Intelligent Credit Scoring: Buildi

Paige Roberts

COVID-19 Is No Worse Than the Flu—And Other Dangerous Myths



Zindi

Eloquent Data: the result of friendship built at a Zindi hackathon

Farooq Mahm... in Analytics Vidh...

Multiprocessing Made Easy(ier) with Databricks

Khachatur Tovma... in SFL Newsr...

Structured Data for leveling-up the SEO of your website

Guillaume Gibon in CITiO

Newsletter #2 : Network in construction use case (March 2019)

Lorela Blaka

Assessing Snowflake: Top-Down Approach