Mining Data Graph

GRAPH GENERATION

Teacher: Le Ngoc Thanh

Email: lnthanh@fit.hcmus.edu.vn

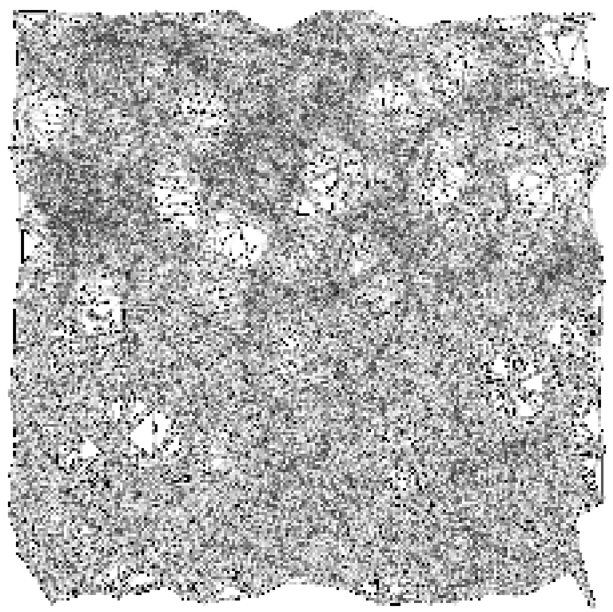
fit@hcmus

Content

- **Graph generation**
- Probability distributions
- NetworkX for graph generation
  - Create and draw simple graphs
  - Functions executed on graphs
  - Synthetic graph generation

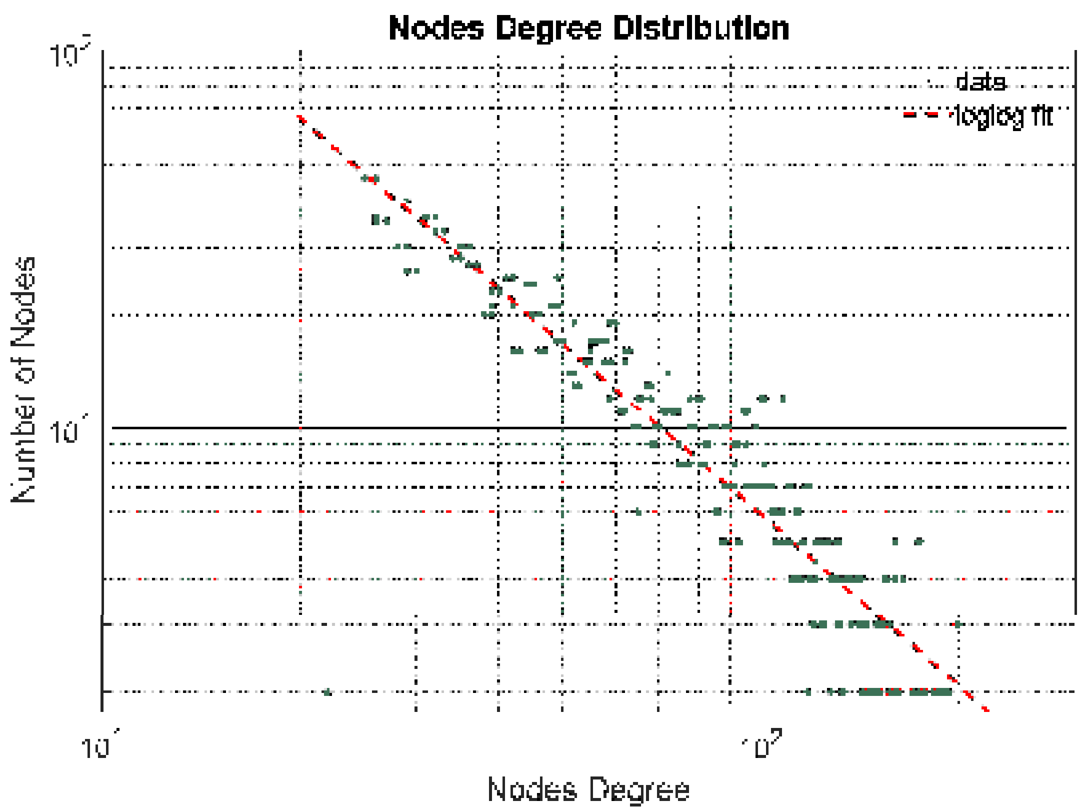
Graph generation

- Graph generative objectives (graph generator) to create synthetic graphs for simulated research.
- Graph requirements are incurred: as close to reality as possible



Synthetic graph

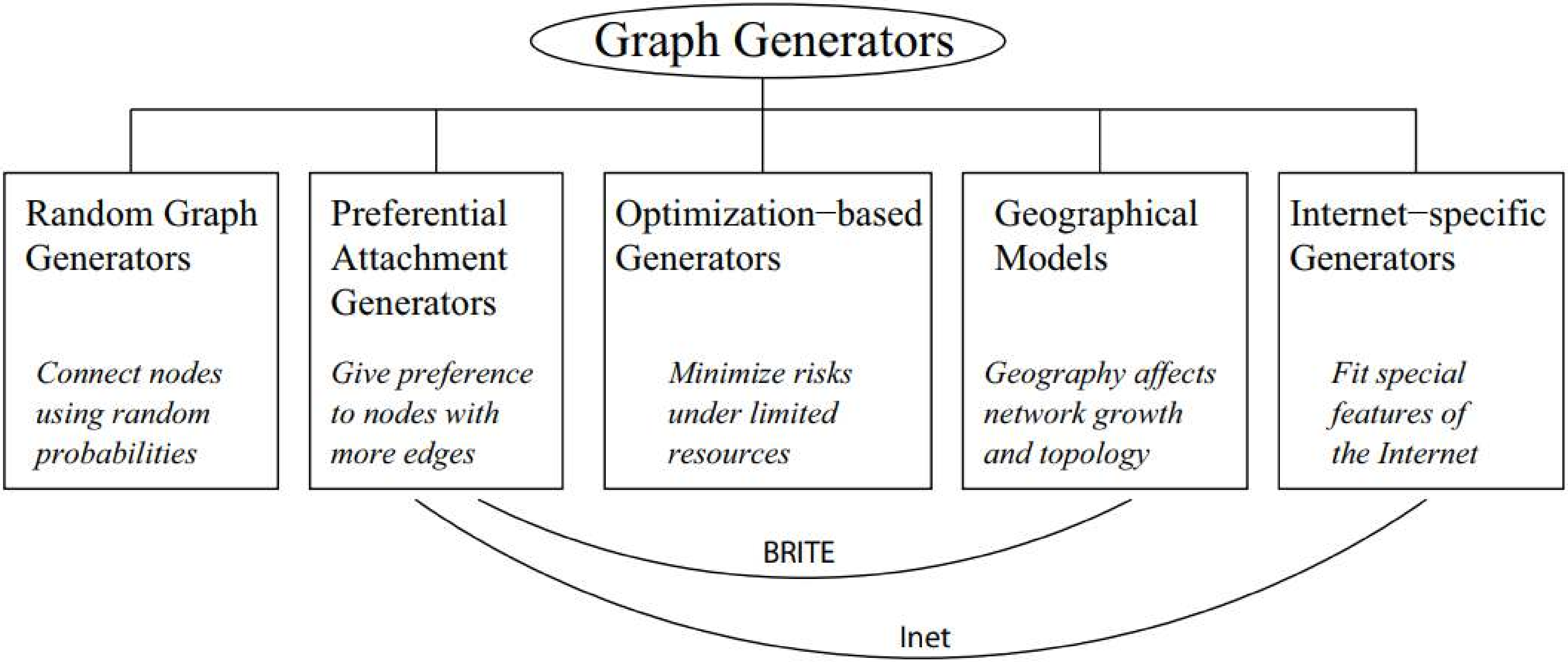
- How does the synthetic graph resemble reality?
    - Large enough size
    - Follow the model rules



Types of composite graphs

- Composite graphs are divided into 5 types:
  - Random graph model
  - Preferential attachment model
  - Optimization-based model
  - Geographical model
  - Internet-specific model

Types of composite graphs



Content

- Graph generation
- Probability distributions
- NetworkX for graph generation
  - Create and draw simple graphs
  - Functions executed on graphs
  - Synthetic graph generation

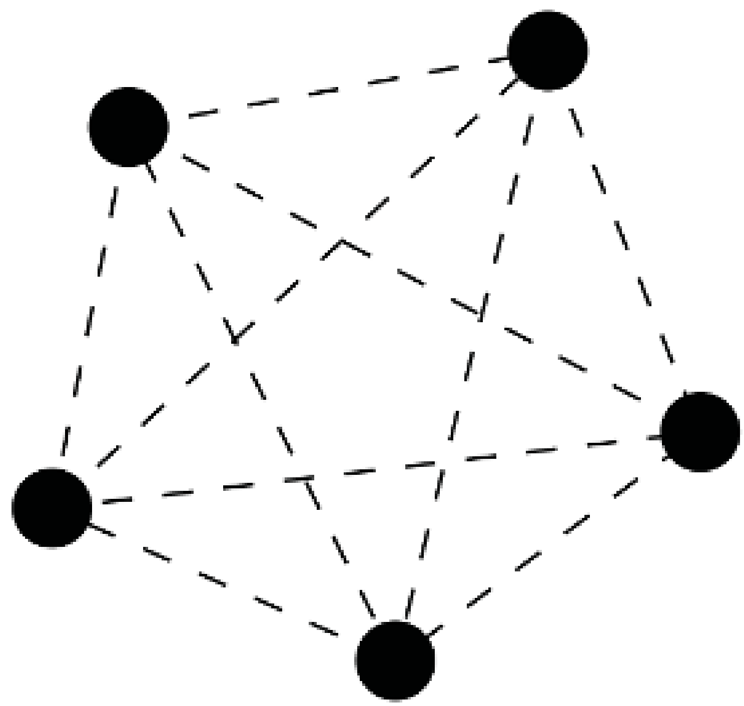
Some related symbols

Symbol	Description
$\mathcal{G}$	A graph with $(\mathcal{V}, \mathcal{E})$ set of nodes and edges
$\mathcal{V}$	Set of nodes for graph $\mathcal{G}$
$\mathcal{E}$	Set of edges for graph $\mathcal{G}$
N	Number of nodes, or $\lvert\mathcal{V}\rvert$
E	Number of edges, or $\lvert\mathcal{E}\rvert$
$e_{i,j}$	Edge between node i and node j
$w_{i,j}$	Weight on edge $e_{i,j}$
w_i	Weight of node i (sum of weights of incident edges)
$\mathbf{A}$	0-1 Adjacency matrix of the unweighted graph
$\mathbf{A}_w$	Real-value adjacency matrix of the weighted graph
$a_{i,j}$	Entry in matrix $\mathbf{A}$
λ_1	Principal eigenvalue of unweighted graph
$\lambda_{1,w}$	Principal eigenvalue of weighted graph
γ	Power-law exponent: $y(x) \propto x^{-\gamma}$

Symbol	Description
k	Random variable: degree of a node
$< k >$	Average degree of nodes in the graph
CC	Clustering coefficient of the graph
$CC(k)$	Clustering coefficient of degree-k nodes
γ	Power-law exponent: $y(x) \propto x^{-\gamma}$

Features of the process of graph phylogeny

- Most graphs are generated by:
  - Select any vertices through some random probability distribution function
  - Connect them to the edges



Probability distributions

- In the stochastic graph method, the probability that a vertex has degree k:
  - Erdös-Rensyi method:

$$p_k = \binom{N}{k} p^k (1-p)^{N-k} \approx \frac{z^k e^{-z}}{k!} \quad \text{with } z = p(N-1)$$

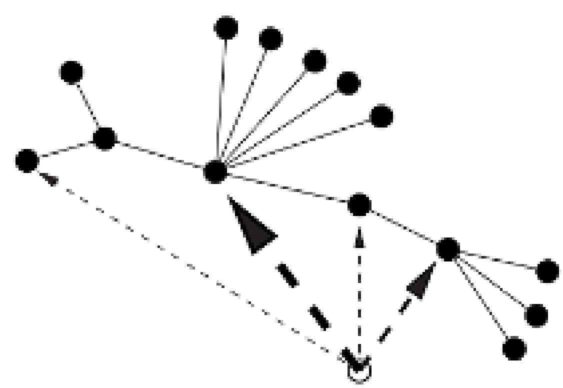
  - PLRG tier distribution:

$$p_k \propto k^{-\beta}$$

Probability distributions

- In the preferred cohesion graph method, the probability that a vertex has degree k:
  - Barabási and Albert method:

$$P(\text{edge to existing vertex } v) = \frac{k(v) + k_0}{\sum_i (k(i) + k_0)}$$

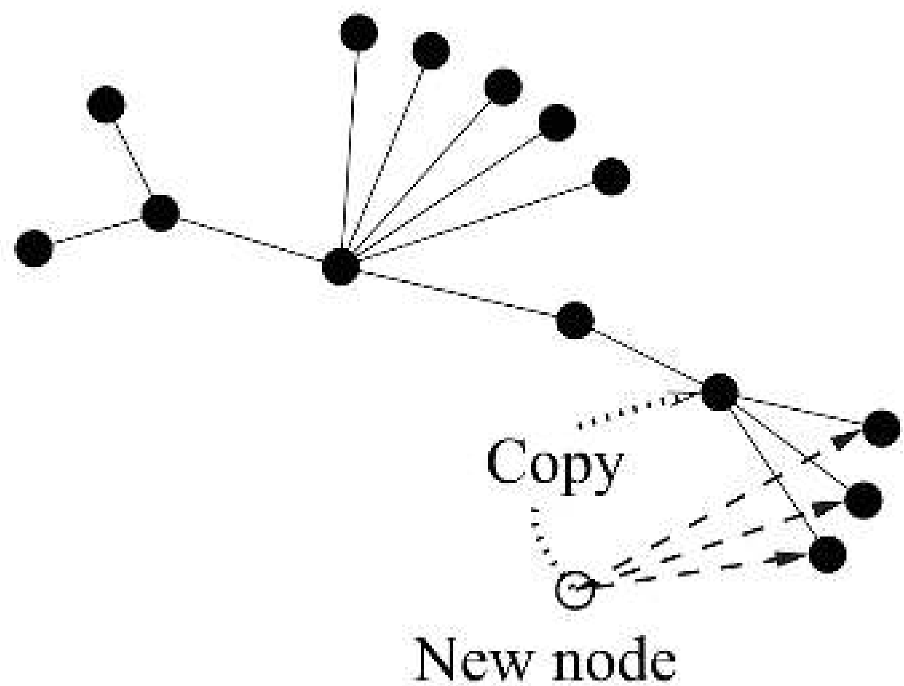


Graph mounting method

- Steps to take:
  - Generate 4 random numbers m(n,n), m(n,e), m(e,n) and m(e,e) according to a probability distribution
  - A vertex is added to the graph after each loop
  - m(n,n) edge is added from the new vertex to itself (advised)
  - m(n,e) edges are added from new vertices to random vertices already existing according to the rule: The higher the inner order the vertex has, the higher the probability of priority being chosen (priority)
  - m(e,n) edges are added from existing vertices to new vertices, existing vertices are randomly selected with probability based on their outer order.
  - m(e,e) edges are added between existing vertices, also based on the inner and outer orders.
- The vertex that has no inner or outer step is discarded.

Probability distributions

- Several variants of the preferred mounting method:
  - New vertices can choose to replicate the edges of an existing vertex.

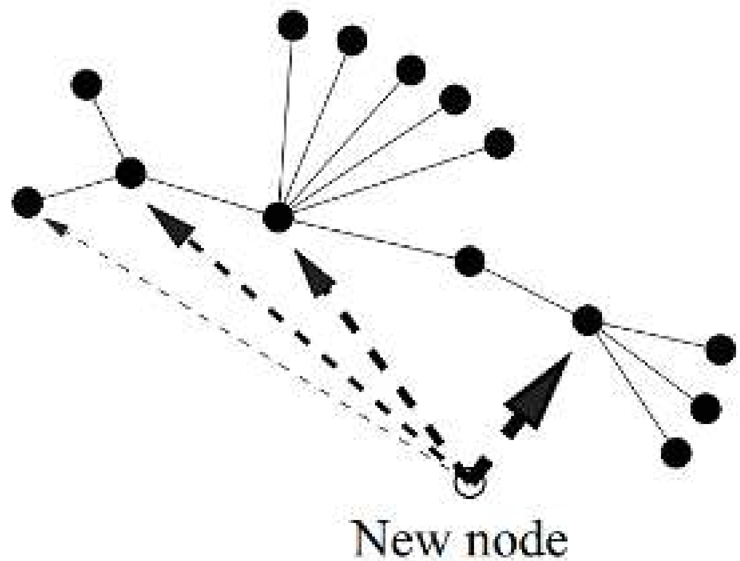


Probability distributions

- In the geograph method, the probability to create an edge
  - Waxman method:

$$P(u,v) = \beta \exp \frac{-d(u,v)}{L\alpha}$$

with $d(u,v)$ is the Euclidean distance between two vertices.

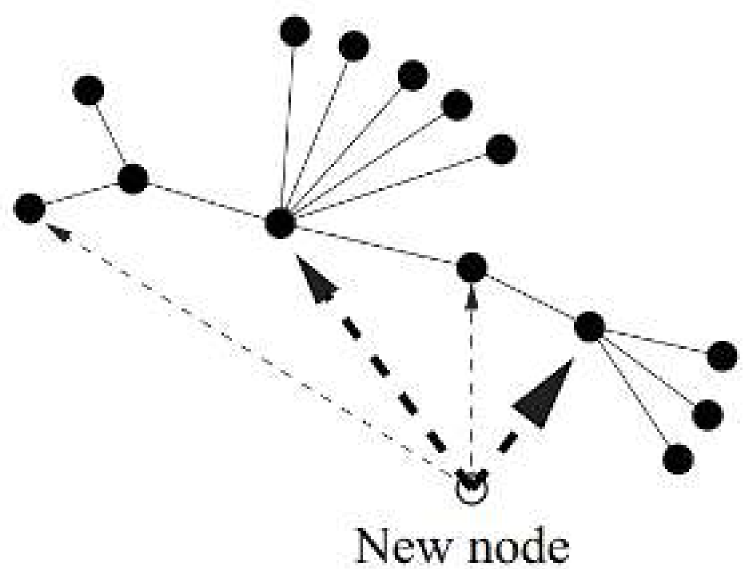


Probability distributions

- In the geograph method, the probability to create an edge
  - Heuristic method:

$$\alpha . d_{ij} + h_j \quad (j < i)$$

with d_{ij} is the distance between two vertices, h_j is the heuristic measure expressing the centerness of the vertex J



Content

- Graph generation
- Probability distributions
- NetworkX for graph generation
  - Create and draw simple graphs
  - Functions executed on graphs
  - Synthetic graph generation

NetworkX

- NetworkX is a Python package for creating, manipulating, and studying the structure, dynamics, and functionality of complex networks.
- Install:

  $ pip install networkx

  or

  $ pip install networkx[all]

Basic usage

```
>>> import networkx as nx
>>> g = nx.Graph()
>>> g.add_node("spam")
>>> g.add_edge(1,2)
>>> print(g.nodes())
[1, 2, 'spam']
>>> print(g.edges())
[(1, 2)]
```

Graph types supported by NetworkX

- Graph: scalar single-graph (piercings allowed)
- DiGraph: Directional Graph Menu (Piercings allowed)
- MultiGraph: multi-scalar graph with parallel edges
- MultiDiGraph: Directional multigraph with parallel edges
-

```
>>> g = nx.Graph()
>>> d = nx.DiGraph()
>>> m = nx.MultiGraph()
>>> h = nx.MultiDiGraph()
```

It is possible to switch from directed graphs to scalars and vice versa:
g.to_undirected()
g.to_directed()

Add vertices

- add_nodes_from(): Add vertices to a graph with arguments that are any ordered set or any object.

```
>>> g = nx.Graph()
>>> g.add_node('a')
>>> g.add_nodes_from( ['b','c','d'])
>>> g.add_nodes_from('xyz')
>>> h = nx.path_graph(5)
>>> g.add_nodes_from(h)
>>> g.nodes()
[0,1,'c','b',4,'d',2,3,5,'x','y','z']
```

Add edge

- add_edge(): Add edges to graphs
  - If a vertex does not exist, it will automatically add the vertex
- add_edges_from(): Add edges from the set

```
>>> g = nx.Graph( [('a','b'),('b','c'),('c'
    ,'a')] )
>>> g.add_edge('a', 'd')
>>> g.add_edges_from([('d', 'c'), ('d', 'b')
    ])
```

Add attributes to vertices

- In the process of adding vertices, we can add properties to vertices.

```
>>> g = nx.Graph()
>>> g.add_node(1,name='Obrian')
>>> g.add_nodes_from([2],name='Quintana'])
>>> g.nodes[1]['name']
'Obrian'
```

Add properties to edges

- Similarly, adding properties to vertices, we can add properties to edges

```
>>> g.add_edge(1, 2, w=4.7 )
>>> g.add_edges_from([(3,4),(4,5)], w =3.0)
>>> g.add_edges_from([(1,2,{'val':2.0})])
# adds third value in tuple as 'weight' attr
>>> g.add_weighted_edges_from([(6,7,3.0)])
>>> g.get_edge_data(3,4)
{'w' : 3.0}
>>> g.add_edge(5,6)
>>> g[5][6]
{}
```

Delete elements

- It is possible to remove buttons and edges from the graph in the same way as adding.

```
>>> G.remove_node(2)
>>> G.remove_nodes_from("spam")
>>> list(G.nodes)
[1, 3, 'spam']
>>> G.remove_edge(1, 3)
```

Viewing some features

- Some features can be viewed

```
>>> list(G.nodes)
[1, 2, 3, 'spam', 's', 'p', 'a', 'm']
>>> list(G.edges)
[(1, 2), (1, 3), (3, 'm')]
>>> list(G.adj[1])  # or list(G.neighbors(1))
[2, 3]
>>> G.degree[1]  # the number of edges incident to 1
2
```

Read graphs from files

- NetworkX also has the function of loading graph data from files.

```
>>> file = 'wiki.txt'
>>> wiki = nx.read_adjlist(file, delimiter
   ='\t', create_using=nx.DiGraph())
```

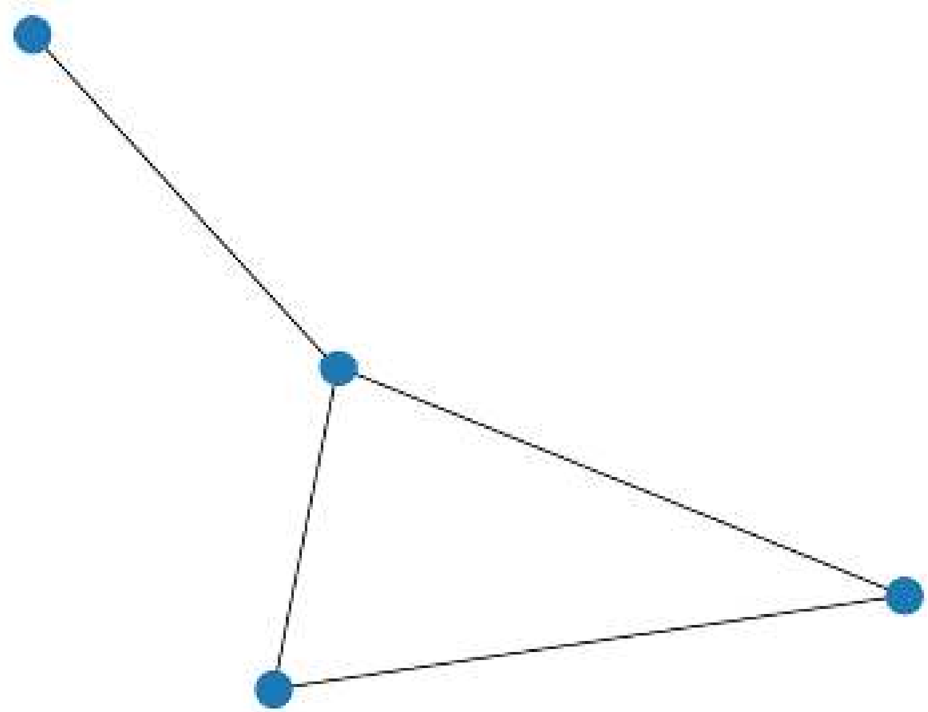
Graphing

- We use the Matplotlib library to draw graphs from network
  - Define a backend to draw on a file or live presentation

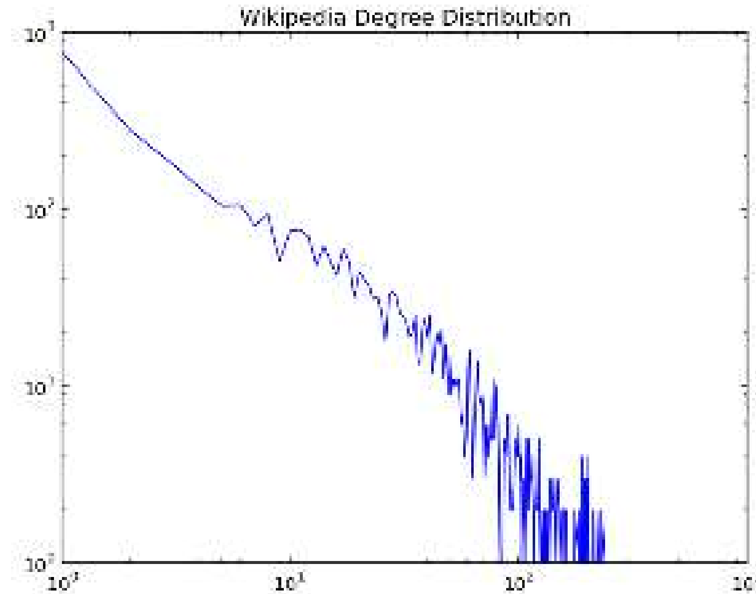
```
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
```

Graphing example

```
G = nx.Graph()
G.add_edges_from([(1,2), (2,3), (1,3),
    (1,4)])
nx.draw(G)
plt.savefig("simple_graph.png")
```



Draw the order distribution




```
degs = {}
for n in wiki.nodes():
  deg = wiki.degree(n)
  if deg not in degs:
    degs[deg] = 0
  degs[deg] += 1
items = sorted(degs.items())
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot([k for (k,v) in items], [v for (k,
   v) in items])
ax.set_xscale('log')
ax.set_yscale('log')
plt.title("Wikipedia Degree Distribution")
fig.savefig("degree_distribution.png")
```

Algorithms on graphs

- Many algorithms on the graph have been implemented by NetworkX (networkx.algorithms)

```
>>> import networkx as nx
>>> help(nx.algorithms)
```

- bipartite
- block
- boundary
- centrality (package)
- clique
- cluster
- components (package)
- core
- cycles
- dag
- distance measures
- flow (package)
- isolates
- isomorphism (package)
- link_analysis (package)
- matching
- mixing
- mst
- operators
- shortest_paths (package)
- smetric

Algorithms on graphs

- Ví dụ

shortest path

```
nx.shortest_path(G,s,t)
```

clustering

```
nx.average_clustering(G)
```

diameter

```
nx.diameter(G)
```

Algorithms on graphs

- Ví dụ

As subgraphs

```
nx.connected_component_subgraphs(G)
```

Operations on Graph

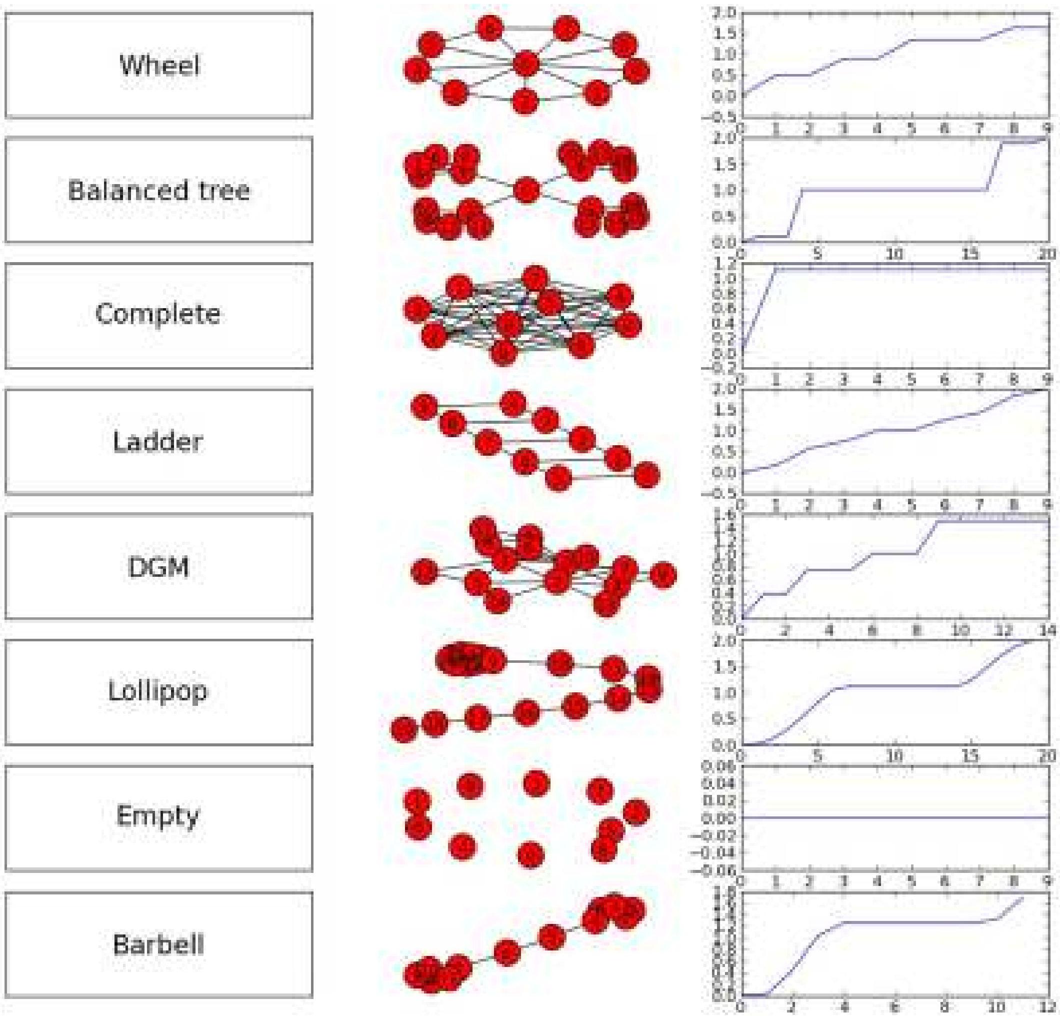
```
nx.union(G,H), intersection(G,H),
    complement(G)
```

k-cores

```
nx.find_cores(G)
```

Graph generation

- NetworkX's graph generator is located in module networkx.generators



Generate a simple graph

Complete Graph

```
nx.complete_graph(5)
```

Chain

```
nx.path_graph(5)
```

Bipartite

```
nx.complete_bipartite_graph(n1, n2)
```

Arbitrary Dimensional Lattice (nodes are tuples of ints)

```
nx.grid_graph([10,10,10,10]) # 4D, 100^4
    nodes
```

Generate random graphs

Preferential Attachment

```
nx.barabasi_albert_graph(n, m)
```

$G_{n,p}$

```
nx.gnp_random_graph(n,p)
```

```
nx.gnm_random_graph(n, m)
```

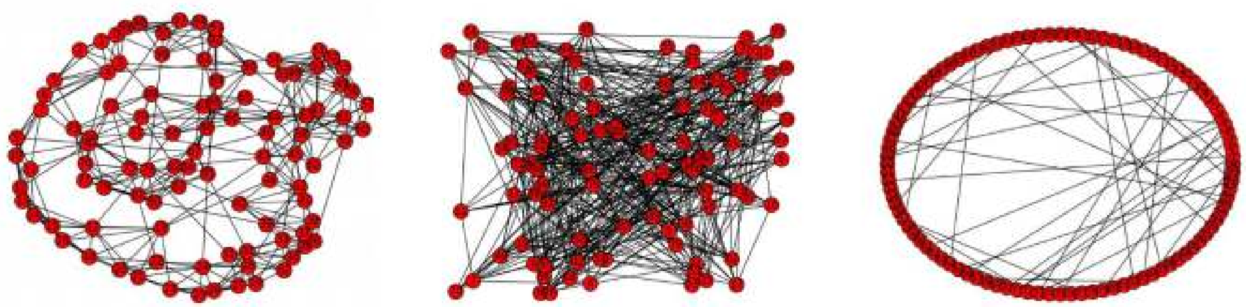
```
nx.watts_strogatz_graph(n, k, p}
```

Graph generation

```
# small famous graphs
>>> petersen = nx.petersen_graph()
>>> tutte = nx.tutte_graph()
>>> maze = nx.sedgewick_maze_graph()
>>> tet = nx.tetrahedral_graph()
```

Plot the generated graph

```
>>> import pylab as plt #import Matplotlib plotting interface
>>> g = nx.watts_strogatz_graph(100, 8, 0.1)
>>> nx.draw(g)
>>> nx.draw_random(g)
>>> nx.draw_circular(g)
>>> nx.draw_spectral(g)
>>> plt.savefig('graph.png')
```



Draw a degree distribution for a generative graph

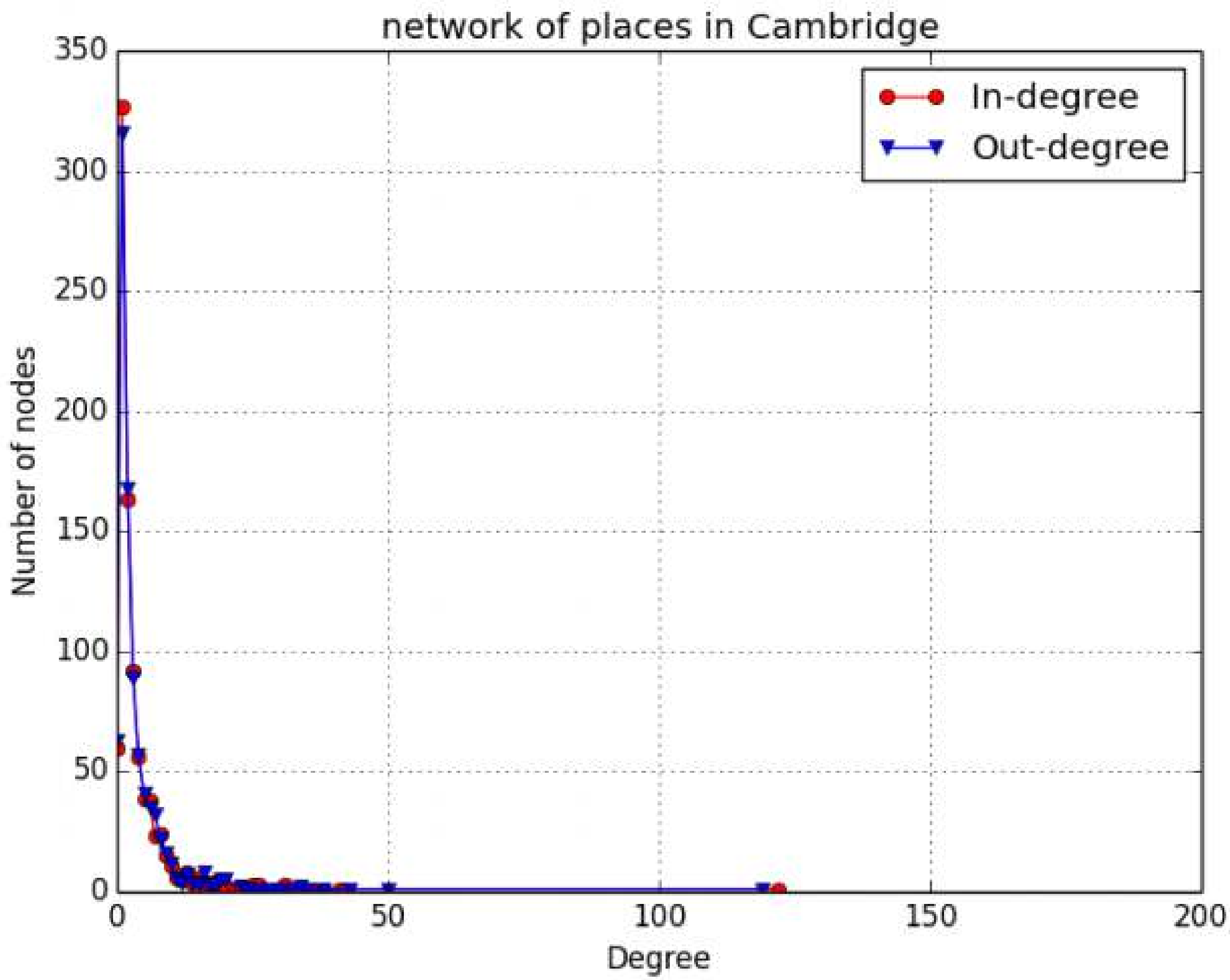
- Calculate in (and out) degrees of a directed graph

```
in_degrees = cam_net.in_degree() # dictionary node:degree
in_values = sorted(set(in_degrees.values()))
in_hist = [in_degrees.values().count(x) for x in in_values]
```

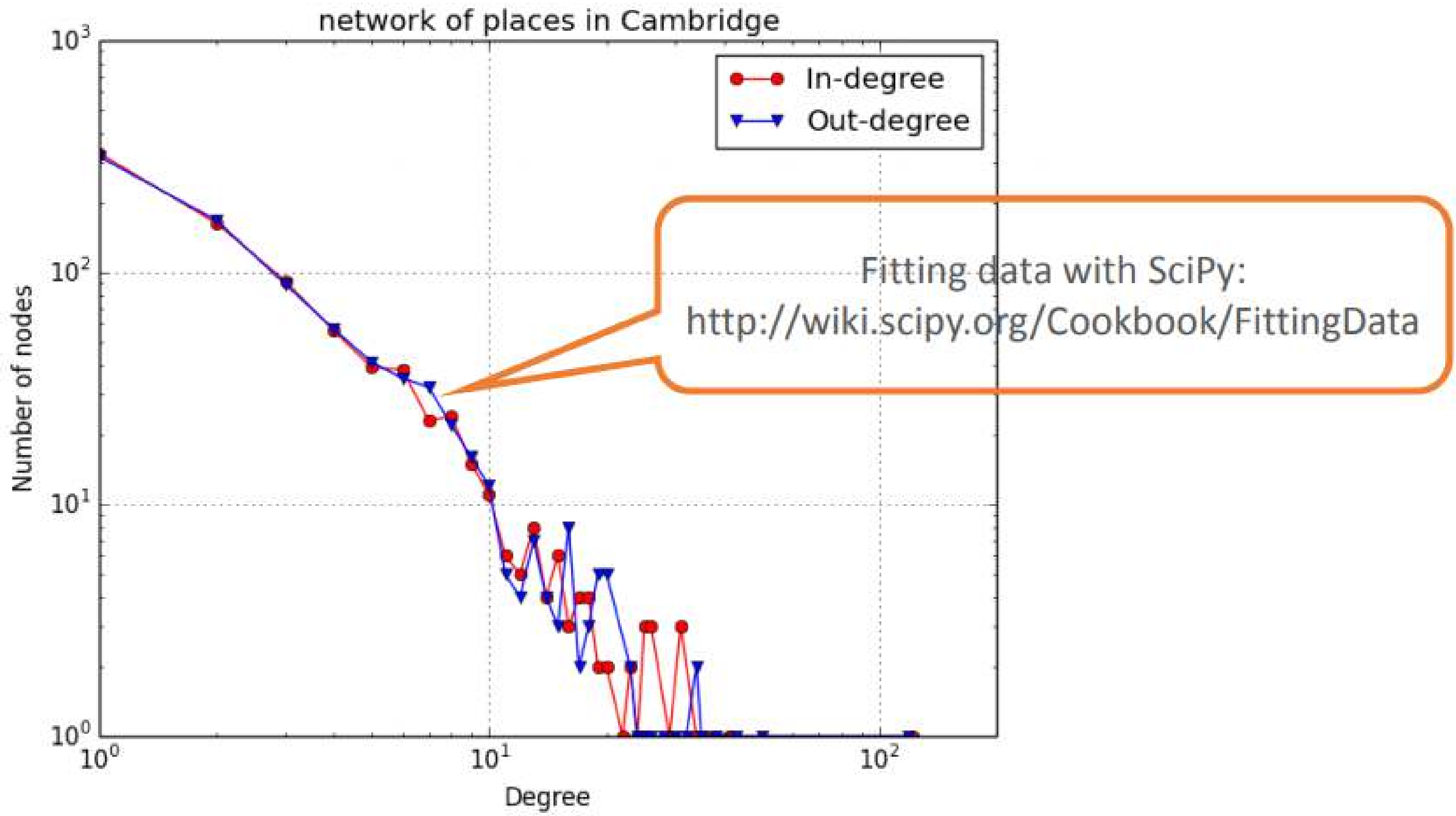
- Then use matplotlib (pylab) to plot the degree distribution

```
plt.figure() # you need to first do 'import pylab as plt'
plt.grid(True)
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('network of places in Cambridge')
plt.xlim([0, 2*10**2])
plt.savefig('./output/cam_net_degree_distribution.pdf')
plt.close()
```

Draw a degree distribution for a generative graph



Draw a degree distribution for a generative graph



Change scale of the x and y axes by replacing

```
plt.plot(in_values,in_hist,'ro-')
```

with

```
plt.loglog(in_values,in_hist,'ro-')
```

References

- Chakrabarti, D. and Faloutsos, C., 2012. Graph mining: laws, tools, and case studies. Synthesis Lectures on Data Mining and Knowledge Discovery, 7(1), pp.1-207.
- https://www.cl.cam.ac.uk/teaching/1314/L109/tutorial.pdf
- https://www.slideshare.net/shankyme/nx-tutorial-basics