

## CS4423 - Networks

Prof. Götz Pfeiffer  
School of Mathematics, Statistics and Applied Mathematics  
NUI Galway

### 7. Growing Graphs

## Lecture 23: Preferential Attachment

In the random graph models we have studied so far, a network is generated on a given number of nodes, and edges are randomly added or modified. Here we introduce and study [preferential attachment \(https://en.wikipedia.org/wiki/Preferential\\_attachment\)](https://en.wikipedia.org/wiki/Preferential_attachment) models, where a network is grown by adding one node at a time, plus some random edges. It turns out that, under suitable circumstances, such a network has a power law degree distribution.

```
In [1]: import numpy as np
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
```

### The Molloy-Reed Criterion

Graphs with a given degree sequence can be analyzed. In particular, a criterion for the existence of a giant component can be formulated in terms of the first and second moments of the degree distribution:

$$\langle k \rangle = \sum_k k p_k, \quad \langle k^2 \rangle = \sum_k k^2 p_k.$$

**\*\*Theorem (Molloy-Reed, 1995/1998).** Suppose  $\underline{p} = (p_0, p_1, \dots)$  is a degree distribution, i.e.,  $\sum_k p_k = 1$  and consider the ensemble of all graphs on  $n$  nodes with degree distribution  $\underline{p}$ . Define

$$Q(\underline{p}) := \sum_k k(k-2)p_k = \langle k^2 \rangle - 2\langle k \rangle.$$

Then: \* if  $Q(\underline{p}) > 0$  almost every graph in the ensemble contains a giant component, \* if  $Q(\underline{p}) < 0$  all components are small.

- For example, in an ER random graph (models  $G(n, m)$  or  $G(n, p)$ ) the moments are related as  $\langle k^2 \rangle = \langle k \rangle^2 + \langle k \rangle$ , whence  $Q(\underline{p}) = \langle k \rangle^2 - \langle k \rangle > 0$  if and only if  $\langle k \rangle > 1$ , as shown previously.
- In a network with power law degree distribution  $p_k \simeq ck^{-\gamma}$  for  $2 \leq \gamma \leq 3$ , the second moment  $\langle k^2 \rangle$  diverges (with  $n \rightarrow \infty$ ) and thus  $Q(\underline{p}) > 0$ : such networks have giant components.

- **Characteristic Path Length.** Scale free networks with  $2 \leq \gamma \leq 3$  are **ultrasmall**:  $L \sim \ln \ln n$  (as opposed to  $L \sim \ln n$  in a small world network.)
- **Clustering Coefficient.** It can be shown that in a scale free network with  $2 \leq \gamma \leq 3$  the clustering coefficient is  $C \approx n^\alpha$ , where  $\alpha = \frac{7-3\gamma}{\gamma-1}$ .

## Citation Networks.

Citation networks are **directed graphs**. In a citation network, the nodes are **scientific publications**, and the directed arcs represent **citations** between papers (where the citing paper points at the cited paper).

- In addition to the network structure, the nodes in a citation network have a natural order, corresponding to their **time of publication**.
- With respect to that order, citations (almost) always **point backwards** in time.
- As a consequence, a citation network is a DAG (**directed acyclic graph**, contains no loops or directed cycles).

**Example.** The Book's website has a data set of citations between the **1656** articles that appeared from 1978 to 2004 in [Scientometrics](https://en.wikipedia.org/wiki/Scientometrics_(journal)) ([https://en.wikipedia.org/wiki/Scientometrics\\_\(journal\)](https://en.wikipedia.org/wiki/Scientometrics_(journal))), a journal devoted to the field of bibliometrics and the "Science of Science". To load it into this notebook, read the file as an edge list for a directed graph ( **Digraph** ), insisting that the nodes are of type **int** . Then copy the edges into an empty directed graph on the nodes **1, 2, ..., 1656**, to have the nodes in their natural order.

```
In [2]: E = nx.read_edgelist("scientometrics.net", create_using=nx.DiGraph, nodetype=int)
        G = nx.DiGraph()
        G.add_nodes_from(range(1, 1656))
        G.add_edges_from(E.edges())
```

Let's check the order and the size of the resulting graph.

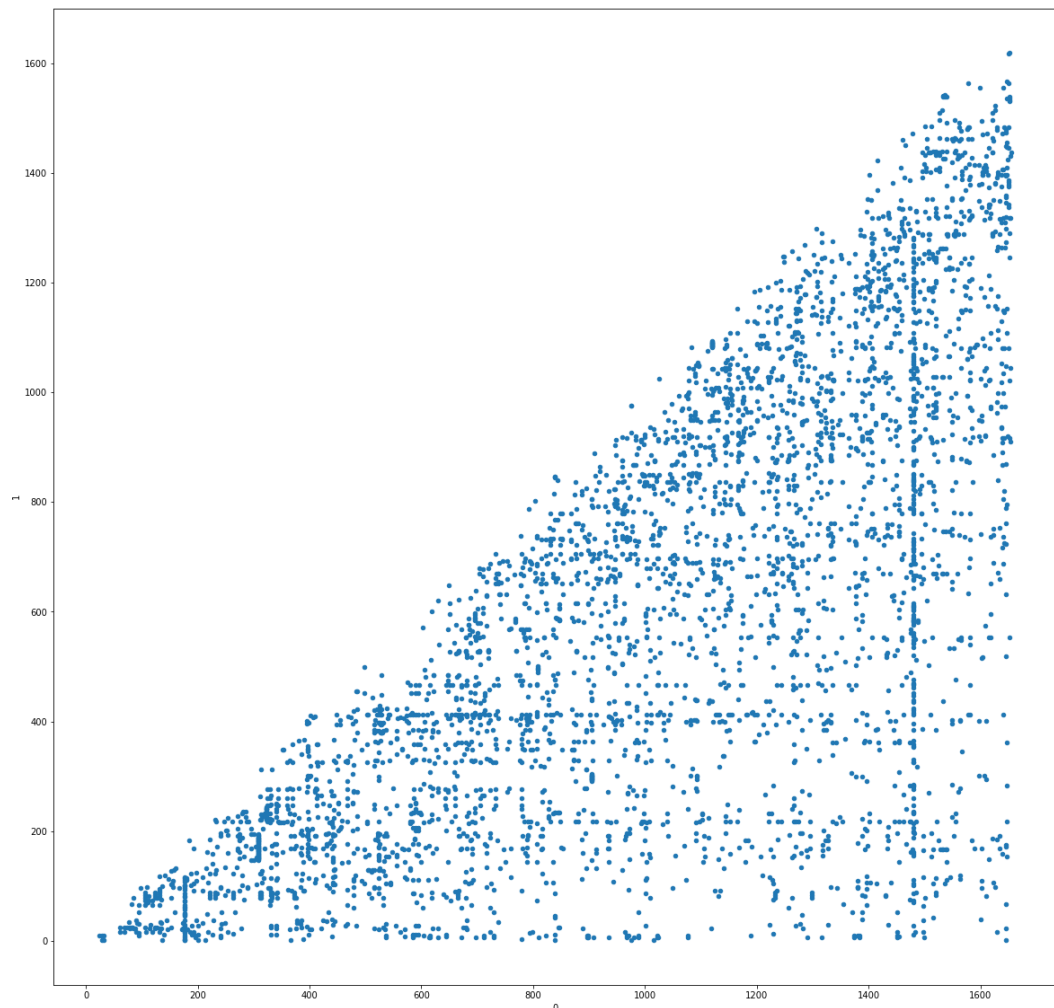
```
In [3]: n, m = G.order(), G.size()
        n, m
```

```
Out[3]: (1655, 4123)
```

A scatter plot of the adjacency matrix reveals the DAG nature of the graph: all its nonzero entries lie below the line  $i = j$ .

```
In [4]: pd.DataFrame(list(G.edges())).plot.scatter(x = 0, y = 1, figsize = (20, 20))
```

```
Out[4]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7249952320>
```



What are the highest degrees? Inwards and outwards?

```
In [5]: max_in = max(dict(G.in_degree()).values())
max_in
```

```
Out[5]: 71
```

```
In [6]: max_out = max(dict(G.out_degree()).values())
max_out
```

```
Out[6]: 156
```

Next, load the time line.

```
In [7]: years = np.loadtxt("scientometrics_paper_year.txt", dtype="int")
years[0]
```

```
Out[7]: array([ 1, 1978])
```

Add the year of publication as attribute to each node.

```
In [8]: for date in years:
        G.nodes[date[0]]['year'] = date[1]
```

How many publications per year? (There may be articles without `year` attribute!)

```
In [9]: counts = {}
        for x in G.nodes():
            year = G.nodes[x].get('year')
            counts[year] = counts.get(year, 0) + 1

        print(counts)

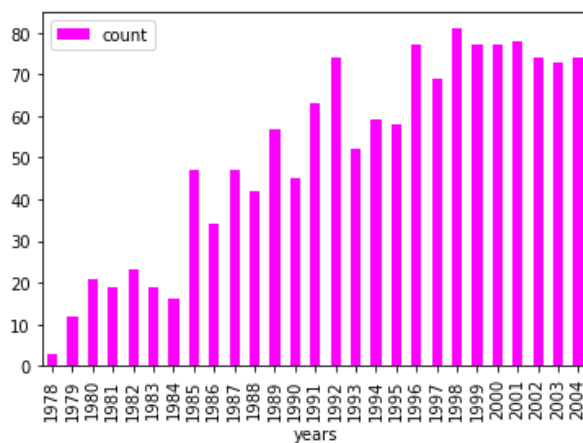
{1978: 3, None: 284, 1979: 12, 1980: 21, 1981: 19, 1982: 23, 1983: 19, 1984:
16, 1985: 47, 1986: 34, 1987: 47, 1988: 42, 1989: 57, 1990: 45, 1991: 63, 199
2: 74, 1993: 52, 1994: 59, 1995: 58, 1996: 77, 1997: 69, 1998: 81, 1999: 77,
2000: 77, 2001: 78, 2002: 74, 2003: 73, 2004: 74}
```

Remove the count of undated articles from the dictionary.

```
In [10]: undated = counts.pop(None, 0)
```

```
In [11]: data = {}
        data['years'] = list(counts.keys())
        data['count'] = list(counts.values())
        pd.DataFrame(data).plot.bar(x = 'years', y = 'count', rot=90, colormap='spring')
```

```
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7248e606d8>
```



Identify node(s) of maximal citation count (in-degree) and show its development over time ...

```
In [12]: hi = [x for x in G if G.in_degree(x) == max_in]
        print(hi)
```

```
[412]
```

```
In [13]: hi = hi[0]
citing = [x for x in G if hi in G[x]]
print(citing)
```

[448, 502, 505, 520, 524, 527, 528, 541, 543, 580, 585, 590, 594, 603, 611, 644, 654, 661, 662, 666, 671, 680, 681, 691, 695, 703, 705, 713, 720, 722, 732, 779, 787, 793, 806, 823, 830, 836, 849, 878, 886, 902, 913, 1003, 1010, 1022, 1023, 1034, 1044, 1127, 1130, 1133, 1149, 1153, 1155, 1181, 1183, 1195, 1200, 1221, 1244, 1266, 1314, 1333, 1396, 1462, 1480, 1510, 1578, 1582, 1641]

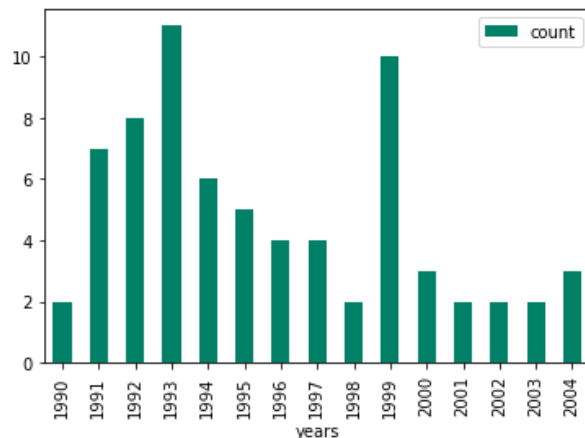
```
In [14]: citing_years = [G.nodes[x].get('year') for x in G if hi in G[x]]
print(citing_years)
```

[1990, 1990, 1991, 1991, 1991, 1991, 1991, 1991, 1991, 1991, 1992, 1992, 1992, 1992, 1992, 1992, 1992, 1992, 1993, 1993, 1993, 1993, 1993, 1993, 1993, 1993, 1993, 1993, 1993, 1993, 1994, 1994, 1994, 1994, 1994, 1994, 1994, 1994, 1995, 1995, 1995, 1995, 1995, 1995, 1996, 1996, 1996, 1996, 1996, 1997, 1997, 1997, 1997, 1997, 1998, 1998, 1999, 1999, 1999, 1999, 1999, 1999, 1999, 1999, 2000, 2000, 2000, 2001, 2001, 2002, 2002, 2003, 2003, 2004, 2004, 2004]

```
In [15]: years = {}
for year in citing_years:
    years[year] = years.get(year, 0) + 1
```

```
In [16]: data = {}
data['years'] = list(years.keys())
data['count'] = list(years.values())
pd.DataFrame(data).plot.bar(x='years', y='count', colormap='summer')
```

Out[16]: <matplotlib.axes.\_subplots.AxesSubplot at 0x7f72495a9be0>

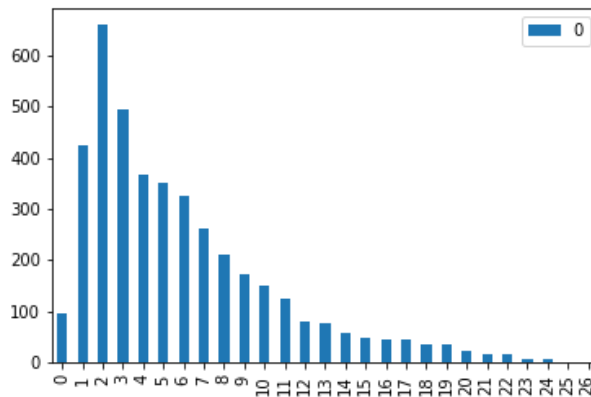


How far back do citations go?

```
In [17]: span = {}
for e in G.edges():
    years = [G.nodes[n].get('year') for n in e]
    if None not in years:
        s = years[0] - years[1]
        span[s] = span.get(s, 0) + 1
```

```
In [18]: all = [x[1] for x in sorted(span.items())]
pd.DataFrame(all).plot.bar()
```

```
Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x7f72494cc470>
```



Go back in time. Recover the state of the citation network at the end of **2003**.

```
In [19]: nodes0 = [x for x in G if G.nodes[x].get('year', 0) < 2004]
nodes1 = [x for x in G if G.nodes[x].get('year', 0) == 2004]
G0 = G.subgraph(nodes0)
G0.order(), G0.size()
```

```
Out[19]: (1581, 3787)
```

Now analyse the references in the articles published in **2004** with respect to the in-degree of the cited article in the **2003** network:

- add citation counter as attribute to nodes in **G0**
- then loop over all nodes and determine citations per in\_degree ...

```
In [20]: for x in G0:
G0.nodes[x]['citations'] = 0

for x in nodes1:
    for y in G[x]:
        if y in G0:
            G0.nodes[y]['citations'] += 1
```

```
In [21]: max_in0 = max(dict(G0.in_degree()).values())
cd = [0 for d in range(max_in0+1)]
hist = [0 for d in range(max_in0+1)]
for x in G0:
    d = G0.in_degree(x)
    hist[d] += 1
    cd[d] += G0.nodes[x]['citations']

print(cd)
print(hist)
```

```
[95, 37, 30, 28, 25, 18, 22, 14, 22, 6, 4, 7, 1, 3, 0, 1, 1, 0, 1, 6, 0, 0, 1,
, 0, 0, 2, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3]
[598, 304, 203, 139, 87, 52, 46, 36, 35, 10, 12, 8, 8, 9, 5, 5, 2, 3, 6, 3, 0,
, 0, 3, 0, 0, 1, 0, 1, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
```

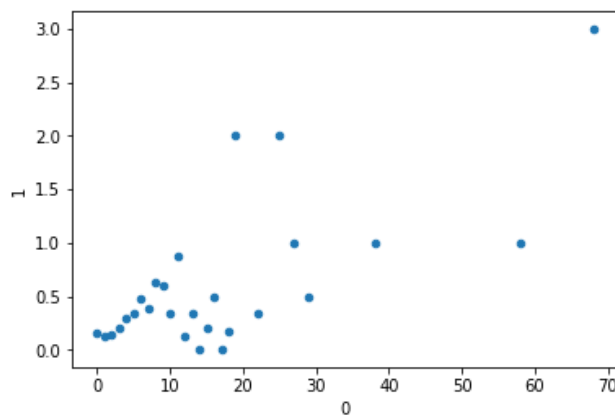
```
In [22]: cd = [(i, cd[i]/v) for i, v in enumerate(hist) if v != 0]
```

```
In [23]: print(cd)
```

```
[(0, 0.1588628762541806), (1, 0.12171052631578948), (2, 0.1477832512315271),
(3, 0.2014388489208633), (4, 0.28735632183908044), (5, 0.34615384615384615),
(6, 0.4782608695652174), (7, 0.3888888888888889), (8, 0.6285714285714286), (9,
0.6), (10, 0.3333333333333333), (11, 0.875), (12, 0.125), (13, 0.3333333333
333333), (14, 0.0), (15, 0.2), (16, 0.5), (17, 0.0), (18, 0.16666666666666666
), (19, 2.0), (22, 0.3333333333333333), (25, 2.0), (27, 1.0), (29, 0.5), (38,
1.0), (58, 1.0), (68, 3.0)]
```

```
In [24]: pd.DataFrame(cd).plot.scatter(x = 0, y = 1)
```

```
Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7248e60b00>
```



Apparently, the number of citations that an article receives is proportional to the number of citations it already has ... the rich get richer.

## Linear Preferential Attachment.

The tendency of new nodes to attach themselves to the most popular nodes, with a probability that is proportional to a node's popularity, has many names, and has been discussed (in a biological context) as far back as 1925. These days, the phenomenon is known as **linear preferential attachment**, after Barabási-Albert, who rediscovered it in their analysis of a portion of the WWW in 1999.

**Definition (BA-Model).** Suppose  $n, a, b$  are integers with  $0 < b \leq a \ll n$ . An  $(n, a, b)$ -BA model is a (simple) graph on  $n$  nodes, constructed as follows. 1. start with a complete graph on  $a$  nodes  $\{1, 2, \dots, a\}$  (at time  $t = 0$ ) 2. for  $t = 1, \dots, n - a - 1$ : \* add new node  $x = a + t$  \* and  $b$  links to old nodes  $i$  with probability

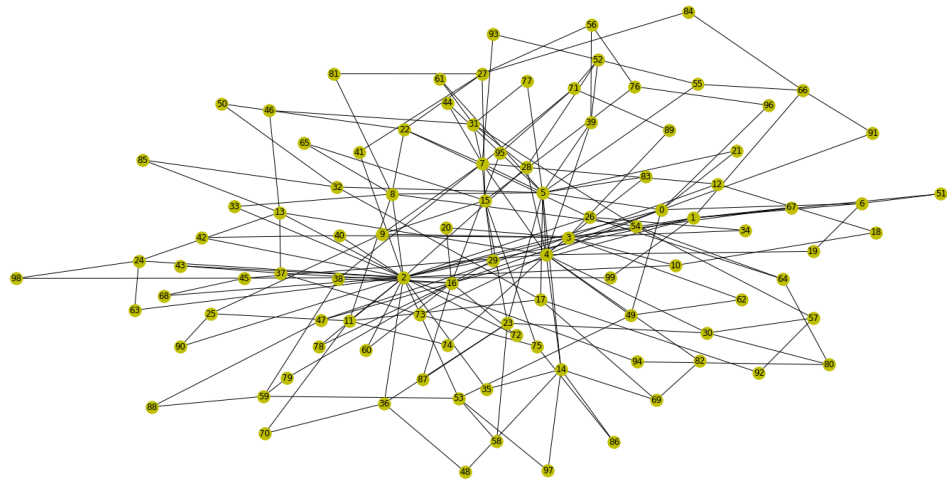
$$p_{x \rightarrow i} = \frac{k_{i,t-1}}{2m_{t-1}},$$

where  $k_{i,t}$  is the degree of node  $i$  at time  $t$  and  $m_t$  is the number of edges at time  $t$ .

In `networkx`, a BA-model can be generated with the function `nx.barabasi_albert_graph(n, b)` (where  $a = b$  in terms of our parameters, and the initial graph is an **empty** graph on the  $a$  nodes  $\{0, 1, \dots, a - 1\}$ ).

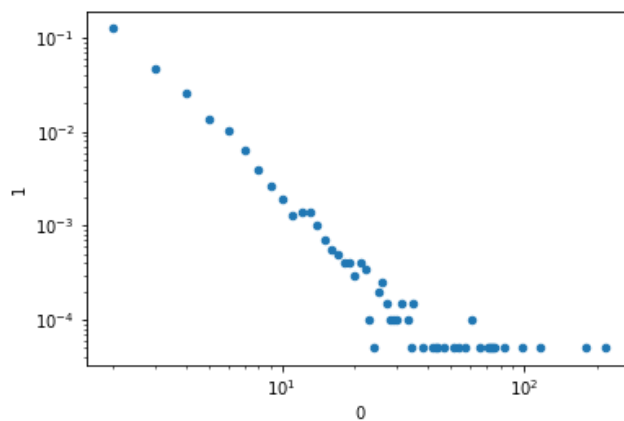
```
In [25]: B = nx.barabasi_albert_graph(100, 2)
```

```
In [26]: plt.figure(figsize=(20,10))
          nx.draw(B, with_labels=True, node_color='y')
```



```
In [27]: B = nx.barabasi_albert_graph(5000, 2)
          hist = nx.degree_histogram(B)
          m2 = 2*B.size()
          data = [(i, v/m2) for i, v in enumerate(hist) if v > 0]
          pd.DataFrame(data).plot.scatter(x = 0, y = 1, loglog=True)
```

```
Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7248e8bf28>
```



**\*\*Fact.\*\*** An  $(n, a, b)$ -BA model (for sufficiently large  $n$ ) has a power law degree distribution  $p_k \simeq ck^{-\gamma}$  with  $\gamma = 3$ .

More precisely,

$$p_k = \frac{2b(b+1)}{k(k+1)(k+2)} \simeq 2b(b+1)k^{-3}.$$

But ...



In [28]: `nx.average_clustering(B)`

Out[28]: 0.007833324294434198

In [ ]: