CS4423 - Networks

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

6. Power Laws and Scale-Free Graphs

Lecture 19: Hubs and Authorities

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

In-Degree vs. Out-Degree

Recall **in-degree** and **out-degree centrality**:

$$c_i^{D^{\text{in}}} = k_i^{\text{in}} = \sum_{j=1}^n a_{ij}, \quad c_i^{D^{\text{out}}} = k_i^{\text{out}} = \sum_{j=1}^n a_{ji},$$

where $A = (a_{ij})$ is the adjacency matrix of a directed graph $G = (X, E)$ ...

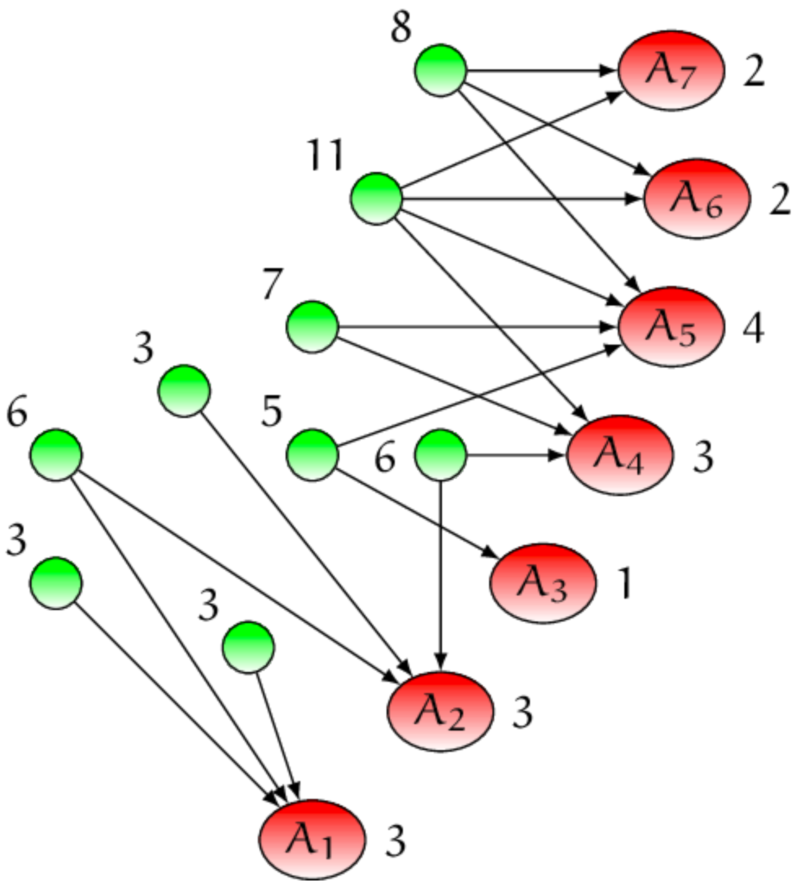
... and the corresponding **eigenvector centralities**:

$$Ac^{E^{\text{in}}} = \lambda c^{E^{\text{in}}}, \quad A^T c^{E^{\text{out}}} = \lambda c^{E^{\text{out}}}.$$

Hub Centrality and Authority Centrality

In a network of nodes connected by directed edges, each node plays two different roles, one as a receiver of links, and one as a sender of links. A first measure of importance, or recognition, of a node in this network might be the number of links it receives, i.e., its **in-degree** in the underlying graph. If in a collection of web pages relating to a search query on the subject of "networks", say, a particular page receives a high number of links, this page might count as an **authority** on that subject, with **authority score** measured by its in-degree.

In turn, the web pages linking to an authority in some sense know where to find valuable information and thus serve as good "lists" for the subject. A high value list is called a **hub** for this query. It makes sense to measure the value of a page as list in terms of the values of the pages it points at, by assigning to its **hub score** the sum of the authority scores of the pages it points at.



Now the authority score of a page could take the hub scores of the list pages into account, by using the sum of the hub scores of the pages that point at it as an updated authority score.

Then again, applying the **Principle of Repeated Improvement**, the hub scores can be updated on the basis of the new authority scores, and so on.

This suggests a ranking procedure that tries to estimate, for each page p, its value as an authority and its value as a hub in the form of numerical scores, $a(p)$ and $h(p)$.

Starting off with values all equal to 1, the estimates are updated by applying the following two rules in an alternating fashion.

> **Authority Update Rule:** For each page p, update $a(p)$ to be the sum of the hub scores of all the pages pointing to it.

> **Hub Update Rule:** For each page p, update $h(p)$ to be the sum of the authority scores of all the pages that it points to.

In order to keep the numbers from growing too large, score vectors should be **normalized** after each step, in a way that replaces h by a scalar multiple $\hat{h} = sh$ so that the entries in $\hat{h}$ add up to 100, say, representing relative percentage values, similarly for a.

After a number of iterations, the values $a(p)$ and $h(p)$ stabilize, in the sense that further applications of the update rules do not yield essentially better relative estimates.

Example. Continuing the example above ...

In [2]:
```
nodes = list(range(1,10)) + ["A%s" % (i+1) for i in range(7)]
print(nodes)
```

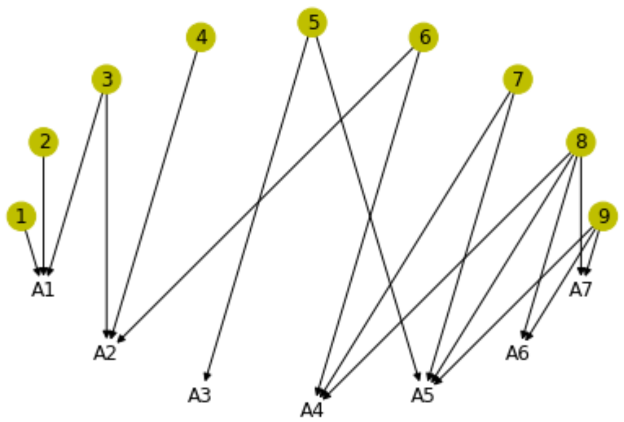
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 'A1', 'A2', 'A3', 'A4', 'A5', 'A6', 'A7']
```

In [3]:
```
edges = [
    (1,"A1"),(2,"A1"),(3,"A1"),(3,"A2"),(4,"A2"),(5,"A3"),
    (5,"A5"),(6,"A2"),(6,"A4"),(7,"A4"),(7,"A5"),(8,"A4"),
    (8,"A5"),(8,"A6"),(8,"A7"),(9,"A5"),(9,"A6"),(9,"A7")
]
```

In [4]:
```
G = nx.DiGraph()
G.add_nodes_from(nodes)
G.add_edges_from(edges)
```

In [5]:
```
pos = nx.circular_layout(G)
for i in [1,2,3,4]:
    j = 10 - i
    pos[i], pos[j] = pos[j], pos[i]
colors = 9 * ['y'] + 7 * ['w']
```

In [6]:
```
nx.draw(G, with_labels=True, node_color=colors, pos=pos)
```



In [7]:
```
A = nx.adjacency_matrix(G)
print(A.todense())
```

```
[[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 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 1 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 0 0 0 0 0 0 1 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 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 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 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 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]]
```

In [8]:
```
AT = A.transpose()
```

In [9]:
```
h = [1 for node in G]
a = [1 for node in G]
print("a = ", a)
print("h = ", h)
```

```
a =  [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
h =  [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

In [10]:
```
a = AT * h
a = 100/sum(a) * a
print("a = ", a[9:])
```

```
a =  [16.66666667 16.66666667  5.55555556 16.66666667 22.22222222 11.11111111
 11.11111111]
```

In [11]:
```
h = A * a
h = 100/sum(h) * h
print("h = ", h[:9])
```

```
h =  [ 5.76923077  5.76923077 11.53846154  5.76923077  9.61538462 11.53846154
 13.46153846 21.15384615 15.38461538]
```

In [12]:
```
aaa = [a]
hhh = [h]
```

In [13]:
```
for i in range(9):
    a = AT * h
    a = 100/sum(a) * a
    print("a = ", a[9:])
    aaa.append(a)
    h = A * a
    h = 100/sum(h) * h
    print("h = ", h[:9])
    hhh.append(h)
```

```
a =  [ 9.6 12.   4.  19.2 24.8 15.2 15.2]
h =  [ 3.35195531  3.35195531  7.54189944  4.18994413 10.05586592 10.89385475
 15.36312849 25.97765363 19.27374302]
a =  [ 5.472103    8.69098712  3.86266094 20.06437768 27.14592275 17.38197425
 17.38197425]
h =  [ 1.92235205  1.92235205  4.97549943  3.05314738 10.89332831 10.10177158
 16.58499812 28.79758764 21.74896344]
a =  [ 3.23741007  6.65467626  3.99833979 20.36524626 28.63862756 18.55285003
 18.55285003]
h =  [ 1.1417976   1.1417976   3.488826    2.3470284  11.51068605  9.52961842
 17.28310725 30.36986435 23.18727432]
a =  [ 2.06677266  5.50149374  4.12131589 20.47380283 29.48514125 19.17573682
 19.17573682]
h =  [ 0.73058966  0.73058966  2.67532917  1.94473951 11.87964724  9.18208542
 17.66013685 31.2171142  23.97976829]
a =  [ 1.46162206  4.87694705  4.19763539 20.51508102 29.94143037 19.50364205
 19.50364205]
h =  [ 0.51731715  0.51731715  2.24343255  1.7261154  12.08296229  8.98709179
 17.85825448 31.66424279 24.4032664 ]
a =  [ 1.15031594  4.54665199  4.2400673  20.5317694  30.1815712  19.67481208
 19.67481208]
h =  [ 0.40740519  0.40740519  2.01768421  1.61027902 12.19104571  8.88197642
 17.9610466  31.89742754 24.62573014]
a =  [ 0.99039832  4.37417366  4.26267053 20.53894254 30.30650876 19.7636531
 19.7636531 ]
h =  [ 0.3508892   0.3508892   1.90061952  1.54973032 12.24754875  8.8264926
 18.01408527 32.01825372 24.74149143]
a =  [ 0.90825735  4.28471463  4.2744909  20.54219544 30.37114618 19.80959775
 19.80959775]
h =  [ 0.32184543  0.32184543  1.84015498  1.51830955 12.27685067  8.79753681
 18.04139121 32.08064659 24.80141933]
a =  [ 0.86605288  4.23848367  4.28062071 20.54373372 30.40449487 19.83330707
 19.83330707]
h =  [ 0.3069187   0.3069187   1.80898623  1.50206753 12.29198691  8.78251915
 18.05543815 32.11280812 24.8323565 ]
```

In [14]:
```
pd.DataFrame(hhh)
```
Out[14]:

	0	1	2	3	4	5	6	7	8	9	
0	5.769231	5.769231	11.538462	5.769231	9.615385	11.538462	13.461538	21.153846	15.384615	0.0	0
1	3.351955	3.351955	7.541899	4.189944	10.055866	10.893855	15.363128	25.977654	19.273743	0.0	0
2	1.922352	1.922352	4.975499	3.053147	10.893328	10.101772	16.584998	28.797588	21.748963	0.0	0
3	1.141798	1.141798	3.488826	2.347028	11.510686	9.529618	17.283107	30.369864	23.187274	0.0	0
4	0.730590	0.730590	2.675329	1.944740	11.879647	9.182085	17.660137	31.217114	23.979768	0.0	0
5	0.517317	0.517317	2.243433	1.726115	12.082962	8.987092	17.858254	31.664243	24.403266	0.0	0
6	0.407405	0.407405	2.017684	1.610279	12.191046	8.881976	17.961047	31.897428	24.625730	0.0	0
7	0.350889	0.350889	1.900620	1.549730	12.247549	8.826493	18.014085	32.018254	24.741491	0.0	0
8	0.321845	0.321845	1.840155	1.518310	12.276851	8.797537	18.041391	32.080647	24.801419	0.0	0
9	0.306919	0.306919	1.808986	1.502068	12.291987	8.782519	18.055438	32.112808	24.832356	0.0	0

In [15]:
```
pd.DataFrame(aaa)
```
Out[15]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	16.666667	16.666667	5.555556	16.666667	22.222222	11.111
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	9.600000	12.000000	4.000000	19.200000	24.800000	15.200
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.472103	8.690987	3.862661	20.064378	27.145923	17.381
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.237410	6.654676	3.998340	20.365246	28.638628	18.552
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.066773	5.501494	4.121316	20.473803	29.485141	19.175
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.461622	4.876947	4.197635	20.515081	29.941430	19.503
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.150316	4.546652	4.240067	20.531769	30.181571	19.674
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.990398	4.374174	4.262671	20.538943	30.306509	19.763
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.908257	4.284715	4.274491	20.542195	30.371146	19.809
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.866053	4.238484	4.280621	20.543734	30.404495	19.833

In terms of matrix algebra this effect can be described as follows.

Spectral Analysis of Hubs and Authorities

Let $M = (m_{ij})$ be the **adjacency matrix** of the directed graph $G = (X, E)$ that is $m_{ij} = 1$ if $x_i \to x_j$ and $m_{ij} = 0$ otherwise, where $X = \{x_1, \ldots, x_n\}$.

We write $h = (h_1, \ldots, h_n)$ for a list of hub scores, with $h_i = h(x_i)$, the hub score of node x_i. Similarly, we write $a = (a_1, \ldots, a_l)$ for a list of authority scores.

The **hub update rule** can now be expressed as a matrix multiplication:

$$h \leftarrow Ma$$

and similarly, the **authority update rule**, using the transpose of the matrix M:

$$a \leftarrow M^T h$$

Applying two steps of the procedure at once yields update rules

$$h \leftarrow MM^T h$$

and

$$a \leftarrow M^T M\, a$$

for h and a, respectively.

In the limit, one expects to get vectors h^* and a^* whose directions do not change under the latter rules, i.e.,

$$(MM^T)h^* = ch^*$$

and

$$(M^T M)a^* = da^*$$

for constants c and d, meaning that h^* and a^* are **eigenvectors** for the matrices MM^T and $M^T M$, respectively.

Using the fact that MM^T and $M^T M$ are **symmetric** matrices ($(MM^T)^T = (M^T)^T M^T = MM^T$), it can indeed be shown that any sequence of hub score vectors h under repeated application of the above update rule converges to a real-valued eigenvector h^* of MM^T for the real eigenvalue c. A similar result exists for any sequence of authority score vectors a.

```
In [ ]:
```