Part 2: Network Analysis - Visualization & Measures

Felix Z. Hoffmann

Jupyter notebook at: https://github.com/Felix11H/spp-workshop-lecture-network-measures (https://github.com/Felix11H/spp-workshop-lecture-network-measures)

Tools for network analysis

NetworkX

- Python based
- community driven
- most accesible tool (pip install..)
- support for directed graphs lacking

Tools for network analysis

graph-tool

- Python interface, algorithms implemented in C++, making use of Boost Graph Library
- dedicated maintainer (Tiago de Paula Peixoto)
- can be difficult to install
- great support for working with directed graphs

Many more ....

- Zenlib, Python, http://zen.networkdynamics.org/ (http://zen.networkdynamics.org/)
- Brain Connectivity Toolbox, MATLAB, https://sites.google.com/site/bctnet/ (https://sites.google.com/site/bctnet/)
- Brain Analysis using Graph Theory (BRAPH), MATLAB, http://braph.org/ (http://braph.org/)
- ...

In [1]:
```
from IPython.core.display import HTML
HTML("""
<style>
.column {
  float: left;
  width: 33.33%;
  padding: 5px;
}

/* Clear floats after image containers */
.row::after {
  content: "";
  clear: both;
  display: table;
}
</style>
""")
```

Out[1]:

In [2]:
```
%matplotlib inline

import networkx as nx
import matplotlib.pyplot as pl
import lib.directed_watts_strogatz as dws

import numpy as np

import graph_tool.all as gt

from lib.nx2gt import nx2gt
```

Introduction to NetworkX

In [4]:
```
%matplotlib inline
```

In [5]:
```
import networkx as nx

g = nx.Graph()
g.add_nodes_from(['A', 'B','C'])
```

In [9]:
```
g.nodes()
```

Out[9]:
```
NodeView(('C', 'A', 'B'))
```

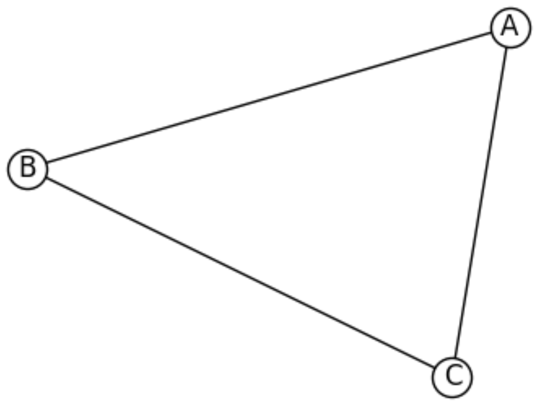
In [10]:
```
g.add_edges_from([('A','B'), ('B','C'),('C','A')])
```

In [11]:
```
g.edges()
```

Out[11]:
```
EdgeView([('C', 'A'), ('C', 'B'), ('A', 'B')])
```

Introduction to NetworkX

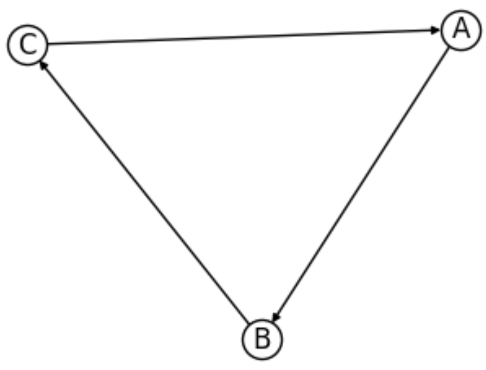
In [12]:
```
pl.axis('off')
nx.draw_networkx(g, node_color = 'white')
```



In [13]:
```
h = nx.DiGraph()

# do not neeed to add nodes explicitly
h.add_edges_from([('A','B'), ('B','C'),('C','A')])
```

Introduction to NetworkX

In [14]:
```
pl.axis('off')
nx.draw_networkx(h, node_color = 'white')
```



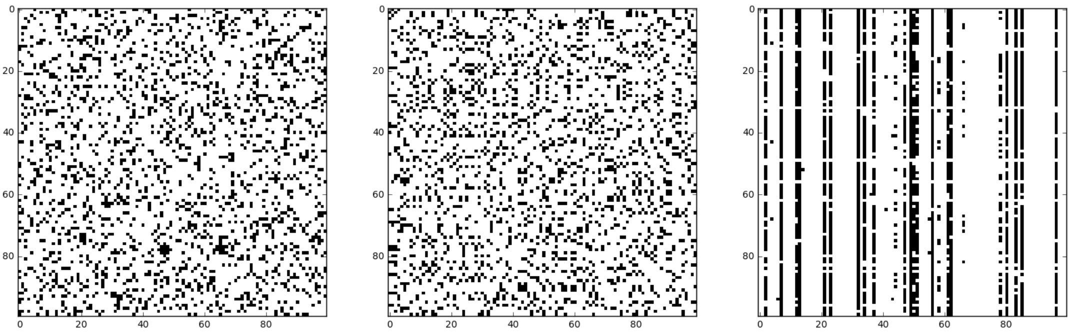
In [3]:

```
def make_graphs(N=50,p=0.2):
    g_edr = nx.gnp_random_graph(N,p, directed=True)
    g_smw = nx.from_numpy_array(dws.watts_strogatz(N, p, 0.1, directed=True
                                create_using=nx.DiGraph())
    #g_scf = nx.scale_free_graph(N)

    x = gt.price_network(N, m=N*p, c=0.1)
    x.save('main.gml')
    g_scf=nx.read_gml('main.gml', label='id')

    return (g_edr, g_smw, g_scf)

def make_graphs_gt(N=50,p=0.2):
    graphs = make_graphs(N)
    gt_s = []
    for g in graphs:
        gt_s.append(nx2gt(g))

    gt_s[-1] = gt.price_network(N, m=N*p, c=0.1)


    gt_s[-1].save('main.gml')

    return gt_s

def shuffle_nodes(g):
    mapping = dict()
    N = g.number_of_nodes()
    xx=np.arange(N)
    np.random.shuffle(xx)
    for i in range(N):
        mapping={**mapping, **{i:xx[i]+N}}
    h = nx.relabel_nodes(g, mapping, copy=False)
    return h
```

Analyzing networks - Visualization

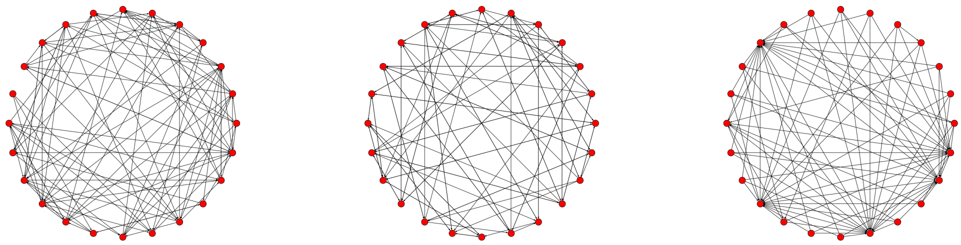
In [5]:

```
nets = make_graphs(N=100)
fig, (axs) = pl.subplots(nrows=1, ncols=3, figsize=(20, 10));
for g,ax in zip(nets,axs):
    g = shuffle_nodes(g)
    A=nx.to_numpy_matrix(g)
    ax.imshow(A, aspect='equal', cmap='Greys',  interpolation='nearest')
```



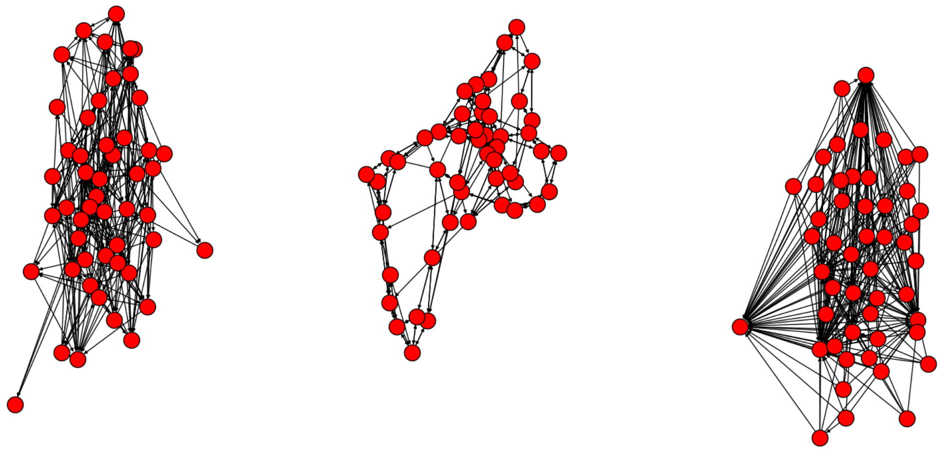
Analyzing networks

In [6]:
```
nets = make_graphs(N=24)
fig, (axs) = pl.subplots(nrows=1, ncols=3, figsize=(40, 20));
for g,ax in zip(nets,axs):
    g=shuffle_nodes(g)
    ax.set_aspect('equal')
    nx.draw_circular(g, ax=ax)
pl.tight_layout()
```



Analyzing networks

In [4]:
```
nets = make_graphs(p=0.1)
fig, (axs) = pl.subplots(nrows=1, ncols=3, figsize=(20, 10));
for g,ax in zip(nets,axs):
    g=shuffle_nodes(g)
    nx.draw(g, ax=ax)
```



Network measures

- global measures pertaining the complete graph
- local measures for a single node (often look at distributions of local node measures or averages)
- regional measures for groups of nodes in a graph

Connection density

$$\text{connection density} = \frac{\text{realized connections}}{\text{possible connections}}$$

In [33]:
```
nets = make_graphs(N=1000)
for g in nets:
    print("N:", nx.number_of_nodes(g), "\t", "density:", nx.density(g))
```

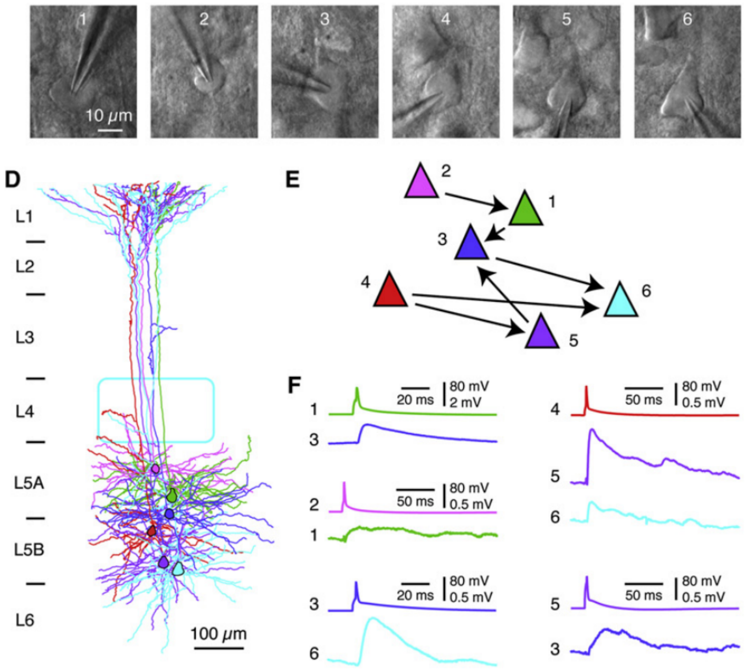
```
N: 1000          density: 0.19946446446446447
N: 1000          density: 0.20014914914914914
N: 1000          density: 0.18008008008008008
```

Connection density in neural circuits

Local cortical circuits

0.05-0.25 -- Song et al. (2005), Lefort et al. (2009), Perin et al. (2011)

Connection density in neural circuits



Lefort et al. (2009)

Connection density in neural circuits

Table 2. Excitatory Synaptic Connectivity and uEPSP Amplitudes in the Mouse C2 Barrel Column

		Presynaptic					
Postsynaptic		L2	L3	L4	L5A	L5B	L6
L2	P (found/tested)	9.3% (88/950)	12.1% (22/182)	12.0% (25/208)	4.3% (9/209)	0.96% (1/104)	0% (0/50)
	mean ± SEM	0.64 ± 0.06 mV	0.71 ± 0.15 mV	0.98 ± 0.24 mV	0.52 ± 0.13 mV	0.21 mV	
	median	0.46 mV	0.59 mV	0.58 mV	0.52 mV		
	range	0.08 – 3.88 mV	0.04 – 2.67 mV	0.07 – 5.54 mV	0.08 – 1.09 mV		
L3	P (found/tested)	5.5% (10/183)	18.7% (96/513)	14.5% (25/172)	2.2% (2/89)	1.8% (3/167)	0% (0/64)
	mean ± SEM	0.44 ± 0.09 mV	0.78 ± 0.07 mV	0.58 ± 0.13 mV	0.67 mV	0.26 ± 0.08 mV	
	median	0.35 mV	0.48 mV	0.35 mV		0.32 mV	
	range	0.09 – 1.02 mV	0.08 – 2.76 mV	0.07 – 3.33 mV	0.15 – 1.19 mV	0.10 – 0.35 mV	
L4	P (found/tested)	0.96% (2/208)	2.4% (4/170)	24.3% (254/1046)	0.7% (2/275)	0.7% (1/137)	0% (0/94)
	mean ± SEM	0.31 mV	0.36 ± 0.09 mV	0.95 ± 0.08 mV	0.48 mV	0.17 mV	
	median		0.31 mV	0.52 mV			
	range	0.18 – 0.45 mV	0.22 – 0.61 mV	0.06 – 7.79 mV	0.22 – 0.74 mV		
L5A	P (found/tested)	9.5% (20/211)	5.7% (5/87)	11.6% (32/276)	19.1% (178/934)	1.7% (3/174)	0.6% (1/160)
	mean ± SEM	0.55 ± 0.10 mV	0.93 ± 0.26 mV	0.54 ± 0.09 mV	0.66 ± 0.06 mV	0.24 ± 0.09 mV	0.08 mV
	median	0.40 mV	1.09 mV	0.38 mV	0.37 mV	0.19 mV	
	range	0.08 – 2.03 mV	0.08 – 1.54 mV	0.06 – 1.98 mV	0.05 – 5.24 mV	0.11 – 0.41 mV	
L5B	P (found/tested)	8.3% (9/108)	12.2% (20/164)	8.1% (11/136)	8.0% (14/175)	7.2% (40/555)	2% (2/100)
	mean ± SEM	0.22 ± 0.04 mV	1.01 ± 0.24 mV	0.88 ± 0.25 mV	0.88 ± 0.36 mV	0.71 ± 0.19 mV	0.30 mV
	median	0.20 mV	0.51 mV	0.44 mV	0.60 mV	0.29 mV	
	range	0.09 – 0.47 mV	0.06 – 4.05 mV	0.07 – 2.61 mV	0.13 – 5.45 mV	0.08 – 7.16 mV	0.12 – 0.48 mV
L6	P (found/tested)	0% (0/50)	0% (0/61)	3.2% (3/93)	3.2% (5/158)	7.0% (7/100)	2.8% (15/532)
	mean ± SEM			2.27 ± 1.72 mV	0.28 ± 0.09 mV	0.49 ± 0.16 mV	0.53 ± 0.19 mV
	median			0.96 mV	0.27 mV	0.43 mV	0.26 mV
	range			0.17 – 5.67 mV	0.06 – 0.58 mV	0.14 – 1.36 mV	0.09 – 3.00 mV

Lefort et al. (2009)

Connection density in neural circuits

Brain area networks

Mouse

- 0.35-0.53 -- Oh et al. (2014), computational model
- 0.73 -- Ypma and Bullmore (2014), re-analysis
- 0.97 -- Gămănuţ et al. (2018)

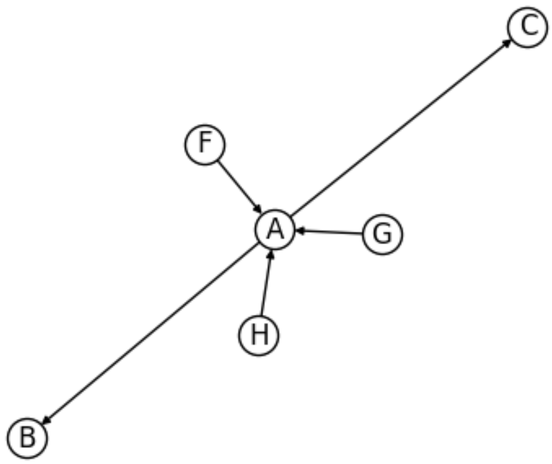
Macaque

- 0.66 --- Markov et al. (2014)

In- and out-degree distributions - local measure

In [9]:
```python
def simple_graph():

    G = nx.DiGraph()

    G.add_edges_from(
        [('A', 'B'), ('A', 'C'), ('H', 'A'), ('G', 'A'), ('F', 'A')])

    val_map = {'A': 1.0,
               'D': 0.5714285714285714,
               'H': 0.0}

    values = [val_map.get(node, 0.25) for node in G.nodes()]

    pl.axis('off')
    nx.draw_networkx(G, node_color = 'white')

    return G
```

In [12]:
```python
g = simple_graph()
```



In [15]:
```python
g.in_degree('A'), g.out_degree('A')
```

Out[15]: (3, 2)

In- and out-degree distributions - local measure

In-degree of a node is the number of incoming connections

Out-degree of a node is the number of outgoing connections

In undirected graphs

$$\text{In-degree} = \text{out-degree}$$

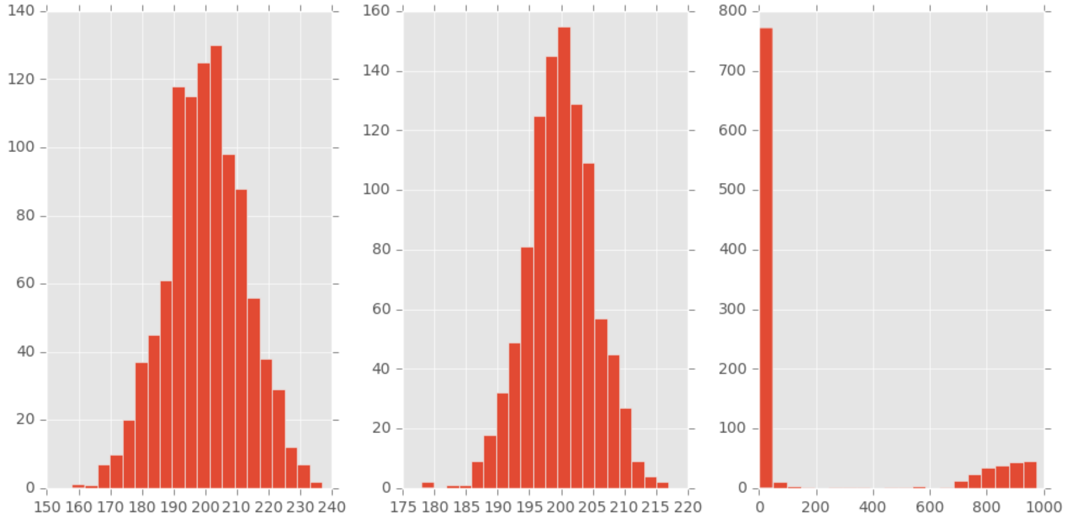
Consistency: Equal number of "heads" and "tails" across graph matches

$$\sum_{v \in V} \deg^{-}(v) = \sum_{v \in V} \deg^{+}(v)$$

In-degree distributions

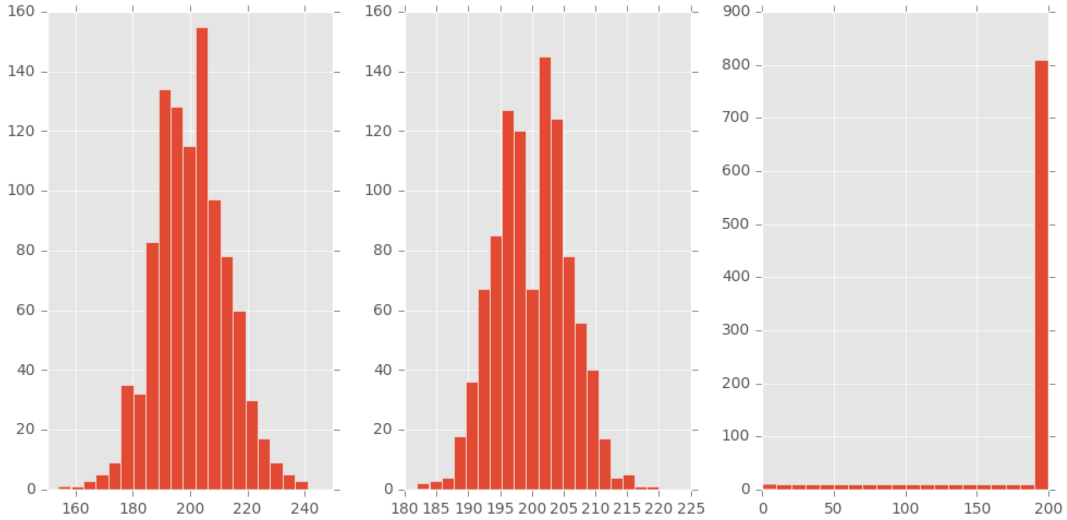
In [8]:
```
pl.style.use('ggplot')
```

In [9]:
```
nets = make_graphs(N=1000)
fig, (axs) = pl.subplots(nrows=1, ncols=3, figsize=(10, 5));
for g,ax in zip(nets,axs):
    ax.hist([x[1] for x in g.in_degree()], bins=20)
pl.tight_layout()
```



Out-degree distributions

In [10]:
```
fig, (axs) = pl.subplots(nrows=1, ncols=3, figsize=(10, 5));
for g,ax in zip(nets,axs):
    ax.hist([x[1] for x in g.out_degree()], bins=20)
pl.tight_layout()
```



Degree distributions in the brain - Theoretical studies

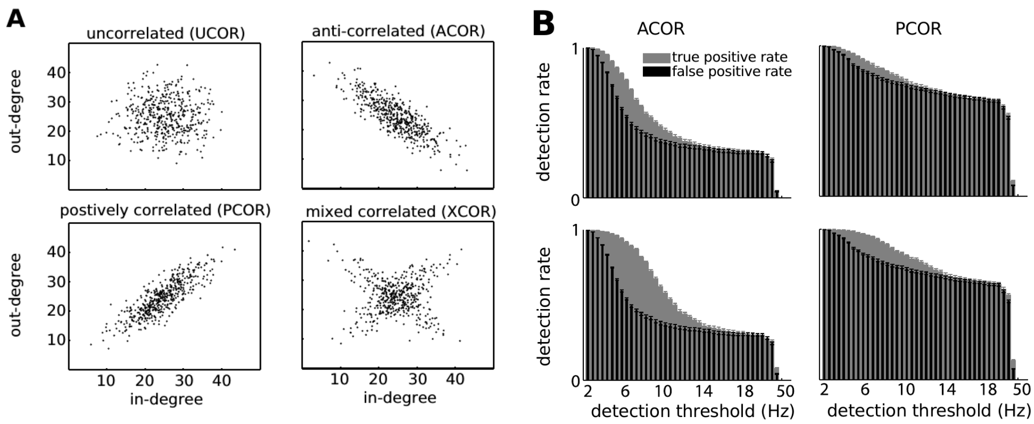
Roxin (2011)

Effect of broadening in-degree and out-degree distributions in recurrent networks

Martens et al. (2017)

Anti-correlated degree distributions increased network stability and had highest performance in detecting stimuli

Degree distributions in the brain - Martens et al. (2017)



B: Stimulation of 3 (top) or 6 (bottom) neurons

Martens et al. (2017)

Clustering - local measure

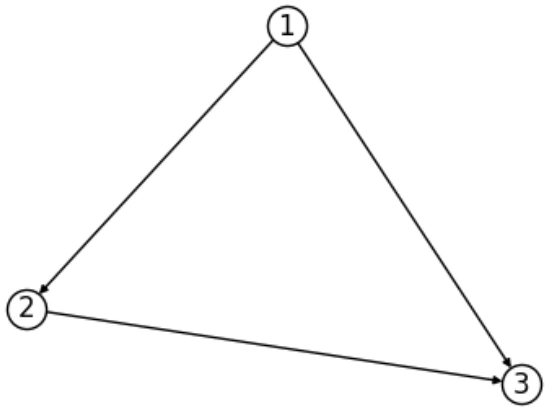
from social network analysis: How many of my friends are friends?

$$c_i = \frac{\text{\# of pairs of } v_i \text{ friends who are friends}}{\text{\# of pairs of } v_i\text{'s friends}}$$

In [4]:
```
def simple_graph1():

    G = nx.DiGraph()

    G.add_nodes_from([1,2,3])

    G.add_edges_from(
        [(1, 2), (1, 3), (2, 3)])


    pl.axis('off')
    nx.draw_networkx(G, node_color = 'white', edge_color='black')

    return nx2gt(G)

def simple_graph2():

    G = nx.DiGraph()

    G.add_nodes_from([1,2,3])

    G.add_edges_from(
        [(1, 2), (1, 3), (2, 3), (3,2)])


    pl.axis('off')
    nx.draw_networkx(G, node_color = 'white', edge_color='black')

    return nx2gt(G)

def simple_graph3():

    G = nx.DiGraph()

    G.add_nodes_from([1,2,3,4])

    G.add_edges_from(
        [(1, 2), (1, 3), (1, 4), (3, 2), (3, 4)])


    pl.axis('off')
    nx.draw_networkx(G, node_color = 'white', edge_color='black')

    return G
```

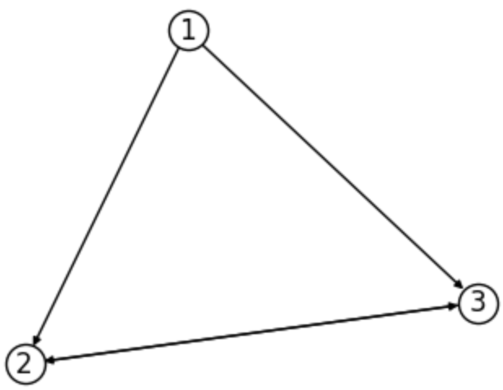
Clustering

In [6]:
```
g = simple_graph1()
list(gt.local_clustering(g, undirected=False))
```
Out[6]: [0.5, 0.0, 0.0]



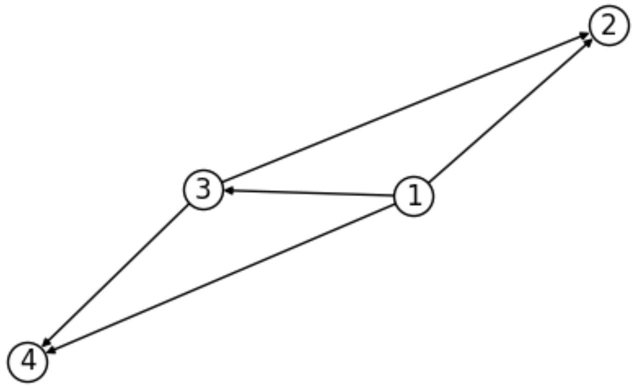
Clustering

In [8]:
```
g = simple_graph2()
list(gt.local_clustering(g, undirected=False))
```
Out[8]: [1.0, 0.0, 0.0]



Clustering

In [37]:
```
g = simple_graph3()
```



In [35]:
```
g=nx2gt(g)
```

In [36]:
```
list(gt.local_clustering(g, undirected=False))
```

Out[36]:
```
[0.3333333333333333, 0.0, 0.0, 0.0]
```

Clustering

The local clustering coefficient c_i is defined as

$$c_i = \frac{|\{e_{jk}\}|}{k_i(k_i - 1)} : v_j, v_k \in N_i, e_{jk} \in E$$

where k_i is the out-degree of vertex i, and

$$N_i = \{v_j : e_{ij} \in E\}$$

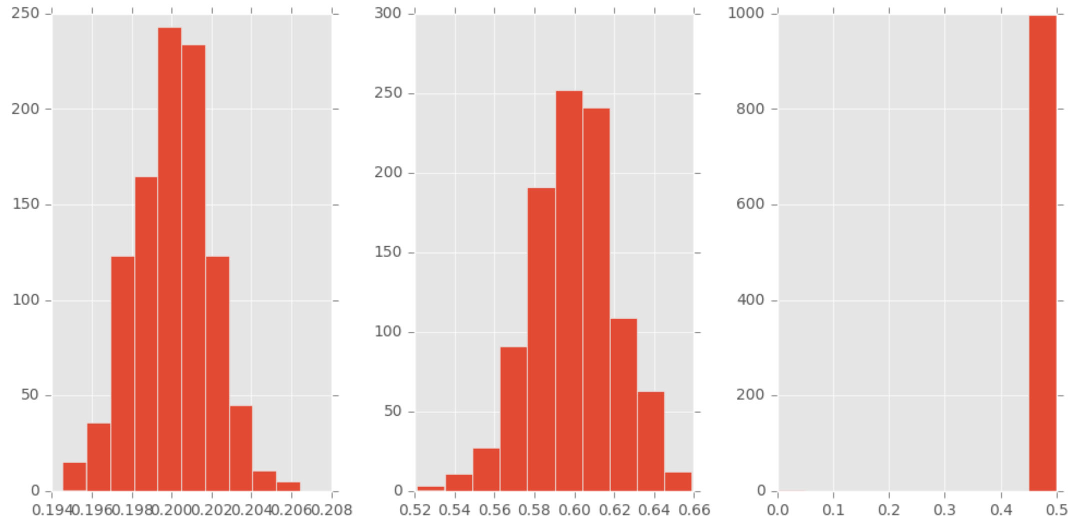
is the set of out-neighbors of vertex i.

Watts and Strogatz (1998)

Clustering

In [42]:
```
nets = make_graphs_gt(N=1000)
fig, (axs) = pl.subplots(nrows=1, ncols=3, figsize=(10, 5));
for g,ax in zip(nets,axs):
    print(gt.global_clustering(g)[0])
    ax.hist(list(gt.local_clustering(g, undirected=False)))
pl.tight_layout()
```

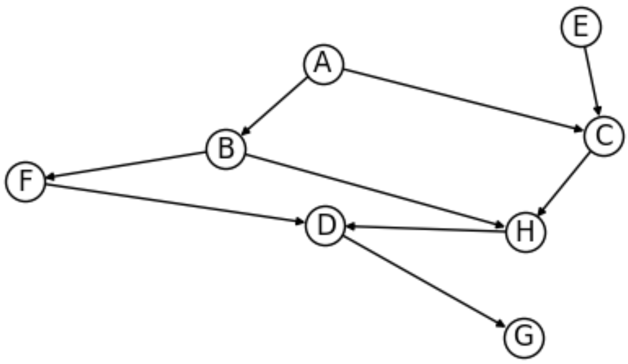
```
0.36015931086427566
0.6516422332814018
0.4867316470895883
```



Shortest Paths

In [50]:
```
def simple_graph():

    G = nx.DiGraph()

    G.add_edges_from(
        [('A', 'B'), ('A', 'C'), ('D', 'G'), ('E', 'C'), ('H', 'D'),
         ('B', 'H'), ('B', 'F'), ('C', 'H'),('F','D')])

    val_map = {'A': 1.0,
               'D': 0.5714285714285714,
               'H': 0.0}

    values = [val_map.get(node, 0.25) for node in G.nodes()]

    pl.axis('off')
    nx.draw_networkx(G, node_color = 'white', edge_color='black')

    return G
```

In [51]:
```
G = simple_graph()
```



In [53]:
```
nx.shortest_path(G, 'A', 'G')
```

Out[53]: ['A', 'B', 'F', 'D', 'G']

In [54]:
```
nx.shortest_path_length(G,'A','G')
```

Out[54]: 4

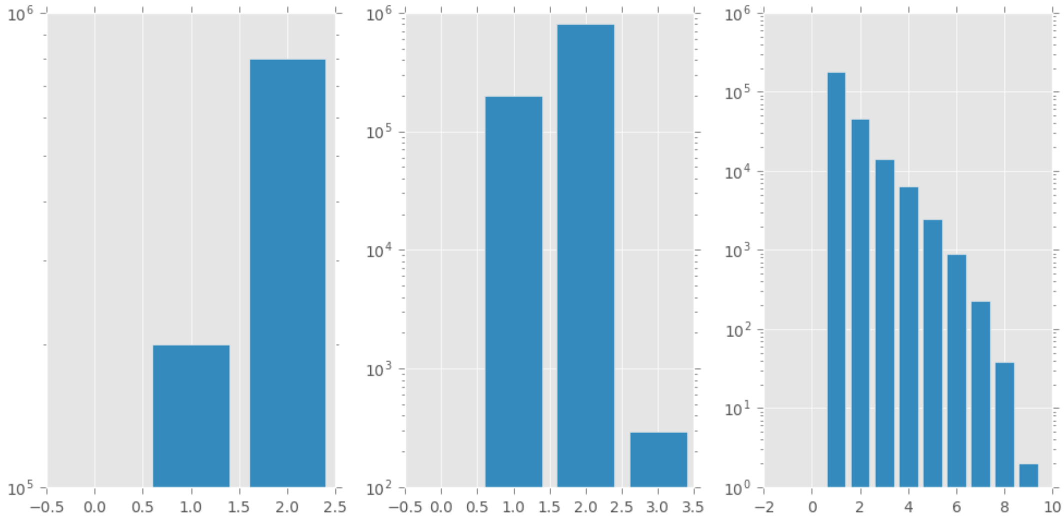
Shortest Paths

In [68]:
```
g=gt.Graph()
g.add_vertex(3);
g.add_edge(g.vertex(0),g.vertex(1))
g.add_edge(g.vertex(1),g.vertex(2))
counts, bins = gt.distance_histogram(g)
print(np.mean((bins[1:]+1)*counts/sum(counts)))
```

```
1.11111111111
```

In [43]:
```
nets = make_graphs_gt(N=1000)
fig, (axs) = pl.subplots(nrows=1, ncols=3, figsize=(10, 5));
for g,ax in zip(nets,axs):
    counts, bins = gt.distance_histogram(g)
    ax.bar(bins[:-1], counts, align='center')
    print(np.sum((bins[:-1])*counts)/np.sum(counts))
    ax.set_yscale('log')
pl.tight_layout()
```

```
1.79968468468
1.80014314314
1.43704912917
```



Shortest Paths - Handling unconnected pairs

several methods suggested:

- does not contribute to average path length
- distance = 0
- distance = N
- distance = ∞

Modularity - regional measure

review: Sporns and Betzel (2016)

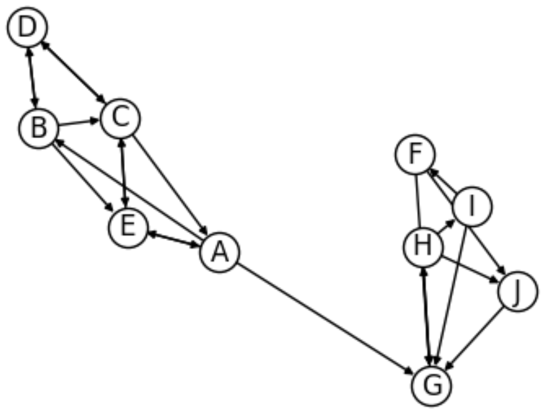
graph tool cookbook on modularity: https://graph-tool.skewed.de/static/doc/dev/demos/inference/inference.html?highlight=partition (https://graph-tool.skewed.de/static/doc/dev/demos/inference/inference.html?highlight=partition)

networkx communities https://networkx.github.io/documentation/stable/reference/algorithms/community.html (https://networkx.github.io/documentation/stable/reference/algorithms/community.html)

graph tool modularity documentation https://graph-tool.skewed.de/static/doc/dev/inference.html#graph_tool.inference.modularity (https://graph-tool.skewed.de/static/doc/dev/inference.html#graph_tool.inference.modularity)

In [7]:
```python
def simple_graph1():

    G = nx.DiGraph()

    G.add_nodes_from(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I','J'])

    G.add_edges_from(
        [('A', 'B'), ('B', 'C'), ('C', 'A'), ('C', 'E'), ('C', 'D'), ('B',
         ('C', 'A'), ('B', 'E'), ('A', 'E'),('F','G'), ('G','H'), ('H','I')
         ('A','G'), ('D', 'B'),
         ('H','J'), ('J','G'), ('H','G'),  ('E','C'), ('E','A'), ('F','J'),

    pl.axis('off')
    nx.draw_networkx(G, node_color = 'white', edge_color='black')

    return nx2gt(G)

def simple_graph2():

    G = nx.DiGraph()

    #G.add_edges_from(
    #   [('A', 'E'), ('A', 'F'), ('A', 'D'), ('B', 'D'), ('D', 'C'),
    #   ('C', 'A'), ('B', 'E'), ('A', 'E'),('F','G'), ('G','H'), ('H','I'),
    #  ('B','I')])

    G.add_nodes_from(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I','J'])

    G.add_edges_from(
        [('A', 'B'), ('B', 'C'), ('C', 'A'), ('C', 'E'), ('C', 'D'), ('B',
         ('C', 'A'), ('B', 'E'), ('A', 'I'),('F','G'), ('G','H'), ('H','I')
         ('A','G'), ('D', 'B'),
         ('H','J'), ('J','G'), ('H','G'),  ('E','C'), ('E','A'), ('F','J'),

    pl.axis('off')
    nx.draw_networkx(G, node_color = 'white', edge_color='black')

    return nx2gt(G)

def simple_graph3():

    G = nx.DiGraph()

    G.add_nodes_from(['A', 'B', 'C', 'D'])

    G.add_edges_from(
        [('A', 'B'), ('B', 'A'), ('C', 'B'), ('C', 'A'), ('B','C'), ('A','C
       ('D', 'E'), ('E', 'D'), ('F', 'E'), ('F', 'D'), ('E','F'), ('D','F')


    pl.axis('off')
    nx.draw_networkx(G, node_color = 'white', edge_color='black')

    return nx2gt(G)

def simple_graph4():

    G = nx.DiGraph()

    G.add_nodes_from(['A', 'B', 'C', 'D', 'E', 'F','G','H','I'])

    G.add_edges_from(
      [('A', 'B'), ('B', 'A'), ('C', 'B'), ('C', 'A'), ('B','C'), ('A','C')
       ('D', 'E'), ('E', 'D'), ('F', 'E'), ('F', 'D'), ('E','F'), ('D','F')
       ('G', 'H'), ('H', 'G'), ('I', 'H'), ('I', 'G'), ('H','I'), ('G','I')


    pl.axis('off')
```

```
In [14]: g = simple_graph1()
```

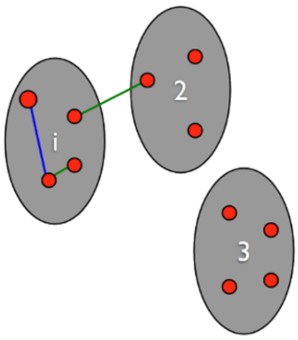


Modularity

probability a random edge would fall into module i

$$Q = \sum_{i=1}^{k} \left(e_{ii} - a_i^2\right)$$

probability edge is in module i



adapted from Carl Kingsford

Modularity

More formally

$$Q = \frac{1}{2E} \sum_r e_{rr} - \frac{e_r^2}{2E}$$

where

- E is the total number of edges
- e_{rs} is the number of edges which fall between vertices in communities s and r, or twice that number if $r = s$
- $e_r = \sum_s e_{rs}$.

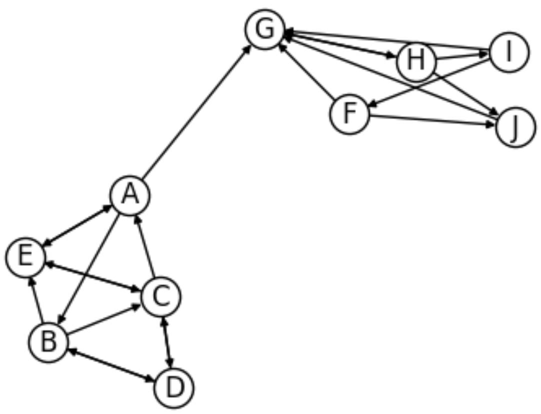
Newman (2006)

Modularity

In [8]:
```
def map_to_vertex(id_list):
    gid = g.vertex_properties['id']
    map_dict = {}
    for v in g.vertices():
        map_dict[int(v)]=gid[v]

    vertices_ids = []
    for idx in id_list:
        for key, item in map_dict.items():
            if idx == item:
                vertices_ids.append(key)
    return [g.vertex(i) for i in vertices_ids]
```

In [97]:
```
g = simple_graph1()
prt1 = g.new_vertex_property('int32_t')

for v in map_to_vertex(['A', 'B', 'C', 'D', 'E']):
    prt1[v]=0
for v in map_to_vertex(['F', 'G', 'H', 'I','J']):
    prt1[v]=1

gt.modularity(g,prt1)
```

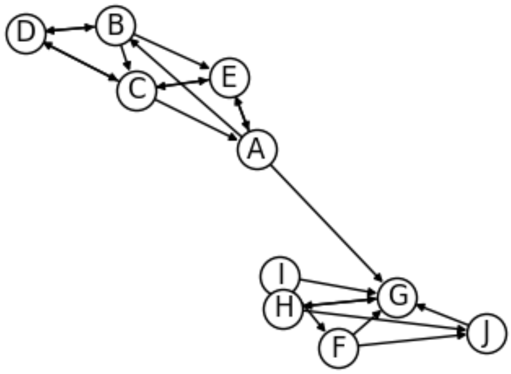
Out[97]: 0.4452479338842975



Modularity

In [85]:
```
g = simple_graph1()
gid = g.vertex_properties['id']
prt2 = g.new_vertex_property('int32_t')

for v in map_to_vertex(['A', 'C', 'G']):
    prt2[v]=0
for v in map_to_vertex(['F', 'H', 'I','J']):
    prt2[v]=1
for v in map_to_vertex(['D', 'B', 'E']):
    prt2[v]=2
gt.modularity(g,prt2)
```

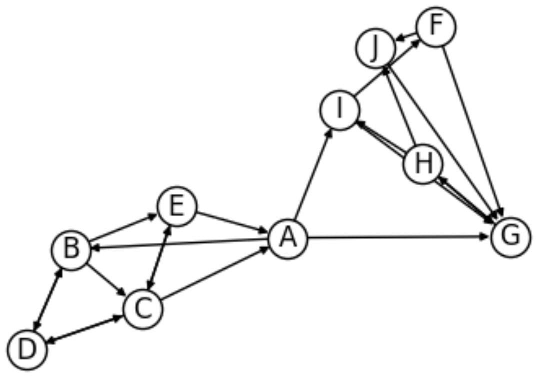
Out[85]: 0.0712809917355372



Modularity

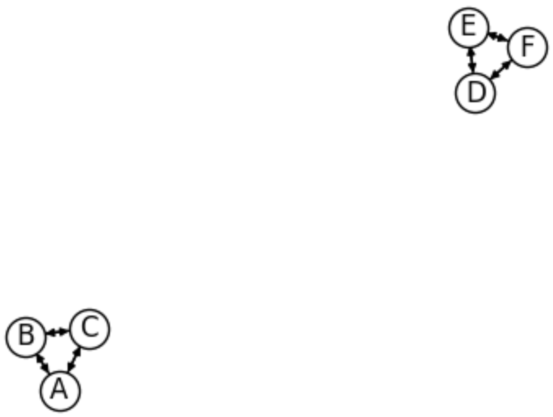
```
In [103]: g = simple_graph2()
          prt1 = g.new_vertex_property('int32_t')

          for v in map_to_vertex(['A', 'B', 'C', 'D', 'E']):
              prt1[v]=0
          for v in map_to_vertex(['F', 'G', 'H', 'I','J']):
              prt1[v]=1

          gt.modularity(g,prt1)
```

```
Out[103]: 0.4049586776859504
```



Modularity - different $Q_{\max}$ for increasing c

```
In [11]: g = simple_graph3()
```



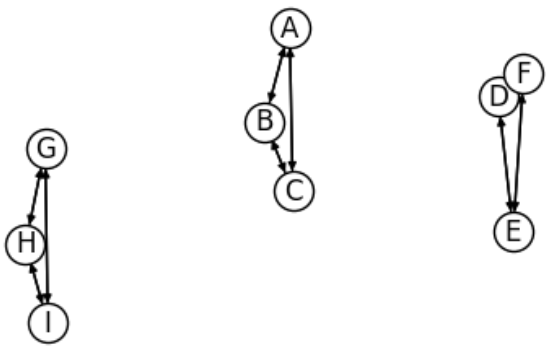
In [14]:
```
prt1 = g.new_vertex_property('int32_t')

for v in map_to_vertex(['A', 'B', 'C']):
    prt1[v]=0
for v in map_to_vertex(['D','E','F']):
    prt1[v]=1
```

In [15]:
```
gt.modularity(g,prt1)
```

Out[15]: 0.5

Modularity - different $Q_{\max}$ for increasing c

In [26]:
```
g = simple_graph4()
```



In [28]:
```
prt1 = g.new_vertex_property('int32_t')

for v in map_to_vertex(['A', 'B','C']):
    prt1[v]=0
for v in map_to_vertex(['D', 'E', 'F']):
    prt1[v]=1
for v in map_to_vertex(['G', 'H', 'I']):
    prt1[v]=2
```

In [29]:
```
gt.modularity(g,prt1)
```

Out[29]: 0.6666666666666666

Modularity - different $Q_{\max}$ for increasing c

$$Q_{\max} = 1 - \frac{1}{c}$$

See also for example Du et al. (2008).

Modularity - optimal partition?

Can try to maximize modularity → **modularity maximization**

Alternatively → fit **stochastic block models**

Stochastic block model parameters:

- The number n of vertices;
- a partition of the vertex set $\{1, \ldots, n\}$ into disjoint subsets $C_1, \ldots, C_r$
- a symmetric $r \times r$ matrix P of edge probabilities.

Then: Any two vertices $u \in C_i$ and $v \in C_j$ are connected by an edge with probability P_{ij}.

Modularity - stochastic block model

Advantage: More flexibility than modularity maximization

Consider

$$P = \begin{bmatrix} 0.8 & 0.2 \\ 0.2 & 0.8 \end{bmatrix}$$

Both modularity maximization and stochastic block model resolve communities.

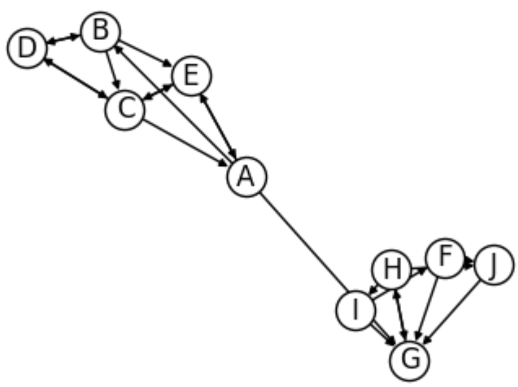
However for

$$P = \begin{bmatrix} 0.8 & 0.2 \\ 0.2 & 0.1 \end{bmatrix}$$

communities detected well by block model but not by modularity maximization (edges out of community more likely for send block)

Modularity - stochastic block model

In [32]:
```
g = simple_graph1()
```

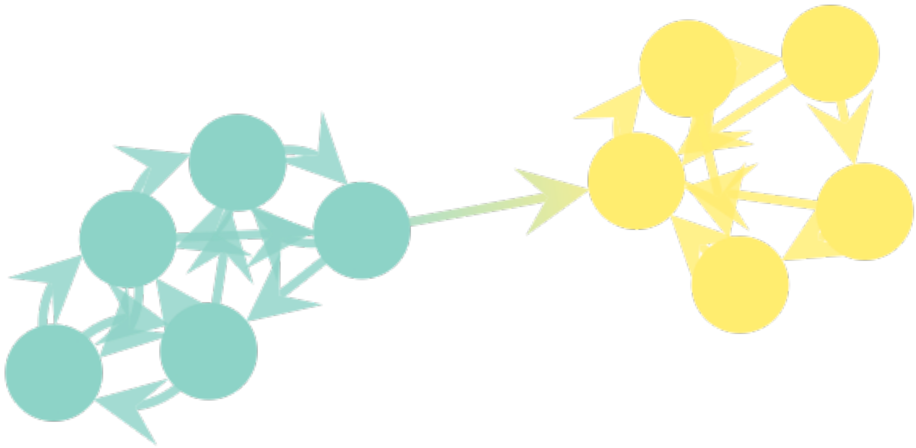


In [41]:
```
state = gt.minimize_blockmodel_dl(g)
b = state.get_blocks()

gid = g.vertex_properties['id']
print([(gid[g.vertex(i)],b[i]) for i in range(10)])
```

```
[('E', 0), ('G', 1), ('B', 0), ('J', 1), ('D', 0), ('C', 0), ('A', 0), ('H
', 1), ('F', 1), ('I', 1)]
```

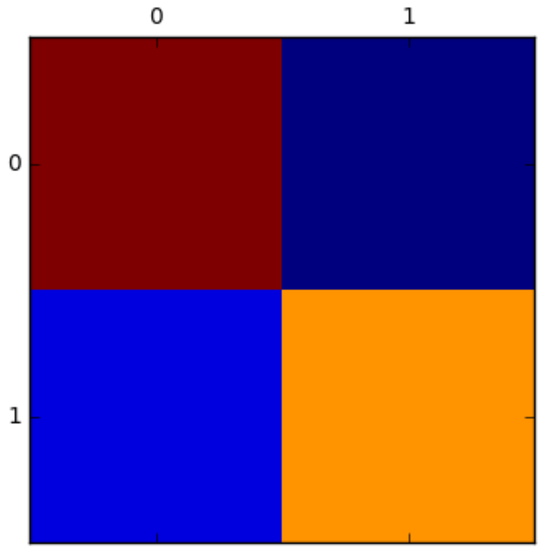
Modularity - stochastic block model

In [39]:
```
state.draw();
```



Modularity - stochastic block model

In [40]:
```
e = state.get_matrix()
pl.matshow(e.todense())
```
Out[40]: <matplotlib.image.AxesImage at 0x7f14d08fb940>



Modularity in brain networks

review: Sporns and Betzel (2016)

C. elegans

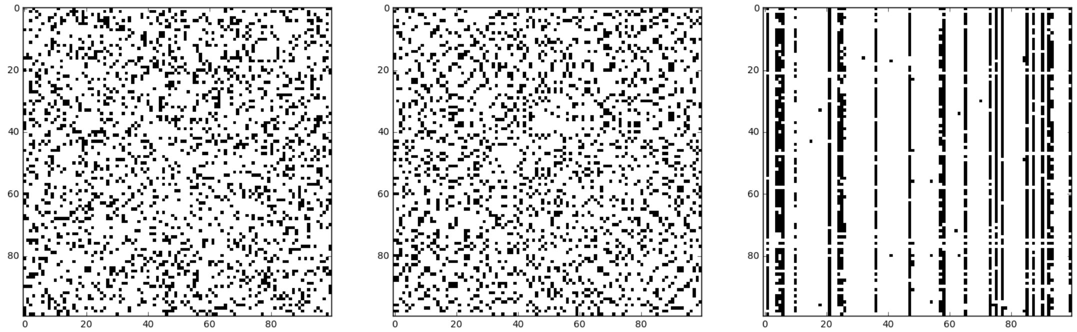
→ resulting communities resemble the functional organization of nervous system (e.g. Jarrel et al. (2012))

Macaque

→ Hilgetag et al. (2000) using optimal set analysis (OSA) before Q modularity was introduced, Harriger et al. 2012 with Q modularity, mostly agreeing with communities identified previously

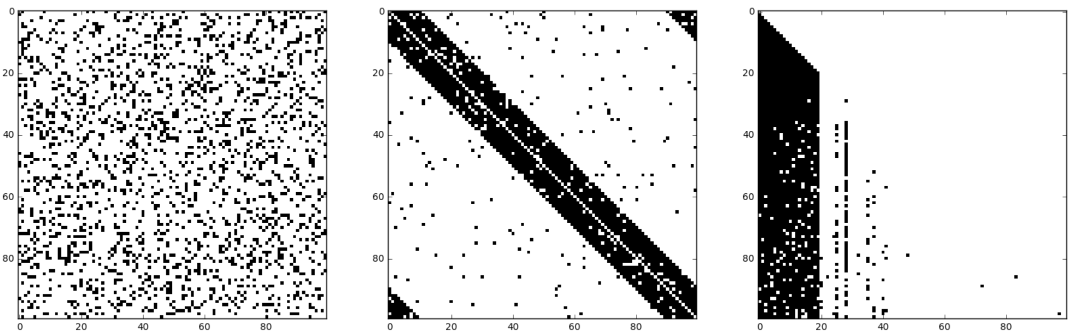
Analyzing networks - Visualization

In [42]:
```
nets = make_graphs(N=100)
fig, (axs) = pl.subplots(nrows=1, ncols=3, figsize=(20, 10));
for g,ax in zip(nets,axs):
    g = shuffle_nodes(g)
    A=nx.to_numpy_matrix(g)
    ax.imshow(A, aspect='equal', cmap='Greys',  interpolation='nearest')
```



Analyzing networks - Visualization

In [43]:
```
nets = make_graphs(N=100)
fig, (axs) = pl.subplots(nrows=1, ncols=3, figsize=(20, 10));
for g,ax in zip(nets,axs):
    #g = shuffle_nodes(g)
    A=nx.to_numpy_matrix(g)
    ax.imshow(A, aspect='equal', cmap='Greys',  interpolation='nearest')
```



References (1/3)

1. Hilgetag, C.-C., Burns, G. A. P. C., O'Neill, M. A., Scannell, J. W. & Young, M. P. Anatomical connectivity defines the organization of clusters of cortical areas in the macaque and the cat. Philosophical Transactions of the Royal Society of London B: Biological Sciences 355, 91–110 (2000).
2. Harriger, L., Heuvel, M. P. van den & Sporns, O. Rich Club Organization of Macaque Cerebral Cortex and Its Role in Network Communication. PLOS ONE 7, e46497 (2012).
3. Jarrell, T. A. et al. The Connectome of a Decision-Making Neural Network. Science 337, 437–444 (2012).
4. Du, H., White, D. R., Ren, Y. & Li, S. A normalized and a hybrid modularity. Draft paper, eclectic. ss. (2008).
5. Newman, M. E. J. Modularity and community structure in networks. PNAS 103, 8577–8582 (2006).

References (2/3)

1. Newman, M. E. J. & Girvan, M. Finding and evaluating community structure in networks. Phys. Rev. E 69, 026113 (2004).
2. Sporns, O. & Betzel, R. F. Modular Brain Networks. Annu. Rev. Psychol. 67, 613–640 (2016).
3. Martens, M. B., Houweling, A. R. & Tiesinga, P. H. E. Anti-correlations in the degree distribution increase stimulus detection performance in noisy spiking neural networks. J Comput Neurosci 42, 87–106 (2017).
4. Markov, N. T. et al. A Weighted and Directed Interareal Connectivity Matrix for Macaque Cerebral Cortex. Cereb Cortex 24, 17–36 (2014).
5. Ypma, R. J. F. & Bullmore, E. T. Statistical Analysis of Tract-Tracing Experiments Demonstrates a Dense, Complex Cortical Network in the Mouse. PLOS Computational Biology 12, e1005104 (2016).

References (3/3)

1. Oh, S. W. et al. A mesoscale connectome of the mouse brain. Nature 508, 207–214 (2014).
2. Watts, D. J. & Strogatz, S. H. Collective dynamics of 'small-world' networks. Nature 393, 440–442 (1998).
3. Lefort, S., Tomm, C., Floyd Sarria, J.-C. & Petersen, C. C. H. The Excitatory Neuronal Network of the C2 Barrel Column in Mouse Primary Somatosensory Cortex. Neuron 61, 301–316 (2009).
4. Roxin, A. The Role of Degree Distribution in Shaping the Dynamics in Networks of Sparsely Connected Spiking Neurons. Front Comput Neurosci 5, (2011).
5. Song, S., Sjöström, P. J., Reigl, M., Nelson, S. & Chklovskii, D. B. Highly Nonrandom Features of Synaptic Connectivity in Local Cortical Circuits. PLoS Biol 3, e68 (2005).
6. Perin, R., Berger, T. K. & Markram, H. A synaptic organizing principle for cortical neuronal groups. PNAS 108, 5419–5424 (2011).

Other - Resources - Exercises?

C Elegans data set in graph tool: "celegansneural" in https://graph-tool.skewed.de/static/doc/collection.html (https://graph-tool.skewed.de/static/doc/collection.html)

Overflow

In [ ]:

```
# overflow

nets = make_graphs_gt(N=50)
#pl.switch_backend('cairo')
#fig, (axs) = pl.subplots(nrows=1, ncols=3, figsize=(20, 10));

#for g,ax in zip(nets,axs):

#fig=pl.figure()
#ax=fig.add_subplot(111)
for k in range(3):
    pos=gt.random_layout(nets[-2],0)
    gt.graph_draw(nets[-2],pos=pos)#, mplfig=ax);

#fig.savefig('new.png')


N = 1000
p = 0.01
g=nx.scale_free_graph(N)

in_degrees = [x[1] for x in g.in_degree()]
xmin,xmax = np.min(in_degrees), np.max(in_degrees)
bins=np.logspace(np.log10(xmin), np.log10(xmax),num=50)
print(bins)
pl.hist(in_degrees, bins=bins)
pl.xscale('log')
pl.yscale('log')
```