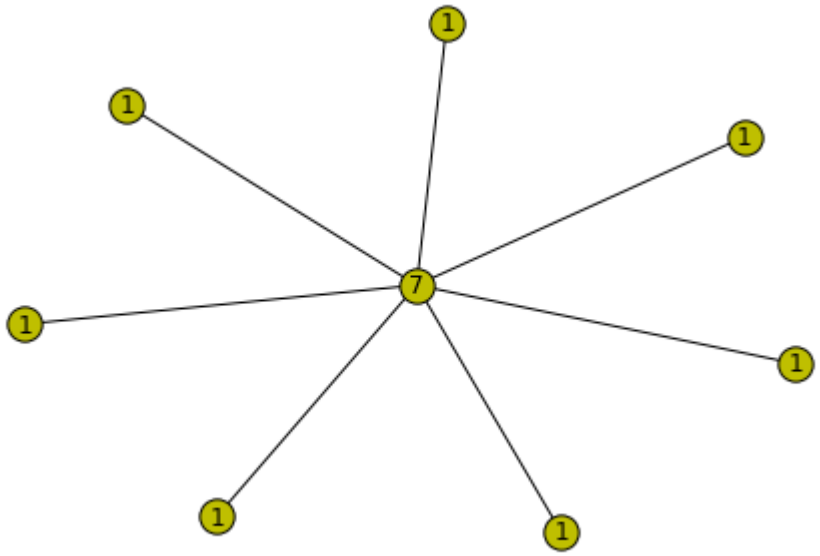
14. Centrality

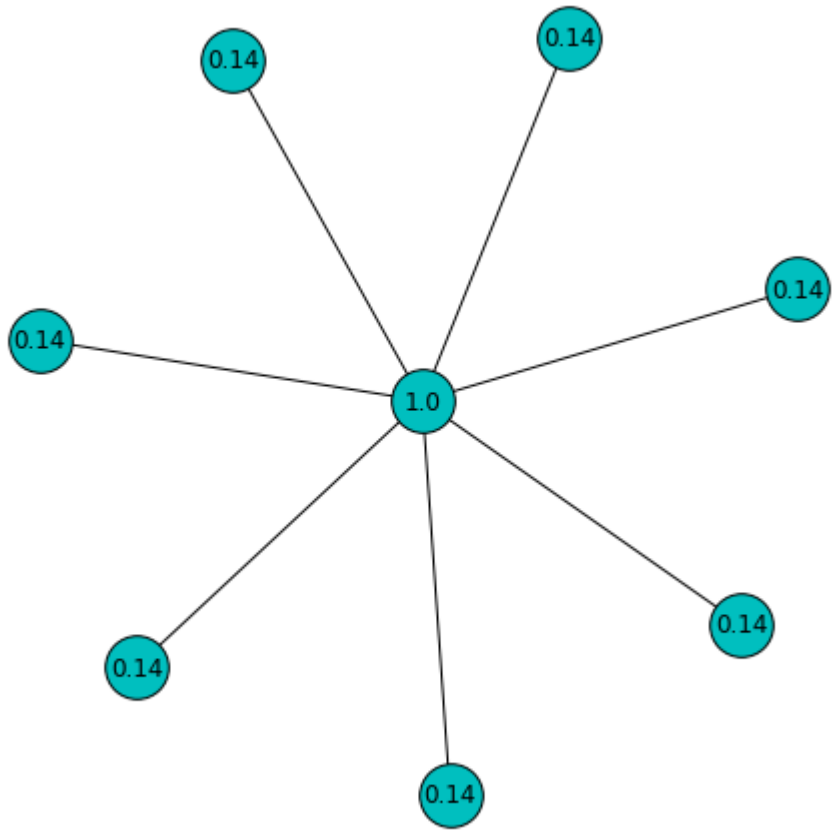
In [1]:
```
%matplotlib inline
import networkx as nx
import matplotlib.pylab as plt
import numpy as np
import netlab as nl

def my_draw(G, pos=None, ncolor='r', nsize=300, nlabels=None, elabels=None):
    if pos == None:
        pos = nx.spring_layout(G)

    if type(nlabels) is str:
        if nlabels=='id':
            nlabels = dict(zip(G.nodes(), G.nodes()))
        else:
            nlabels = nx.get_node_attributes(G, nlabels)

    nx.draw_networkx(G, pos=pos, labels=nlabels, node_color=ncolor, node_siz
#    nx.draw_networkx_edges(G, pos=pos, node_color=ncolor, node_size=nsize)

    if elabels != None:
        nx.draw_networkx_edge_labels(G, pos=pos, edge_labels=elabels)
    return

def roffdict(X, dec=2):
    C = map(lambda a:round(X[a],dec), X)
    return dict(zip(X.keys(), C))

def roff(X, dec=2):
    return map(lambda a: round(a, dec), X)
```

Degree Centrality

In [2]:
```
G = nx.read_gml('Graphs/C01.gml')
nl.draw_atlas(G, node_color='y', labels=dict(G.degree()))
```

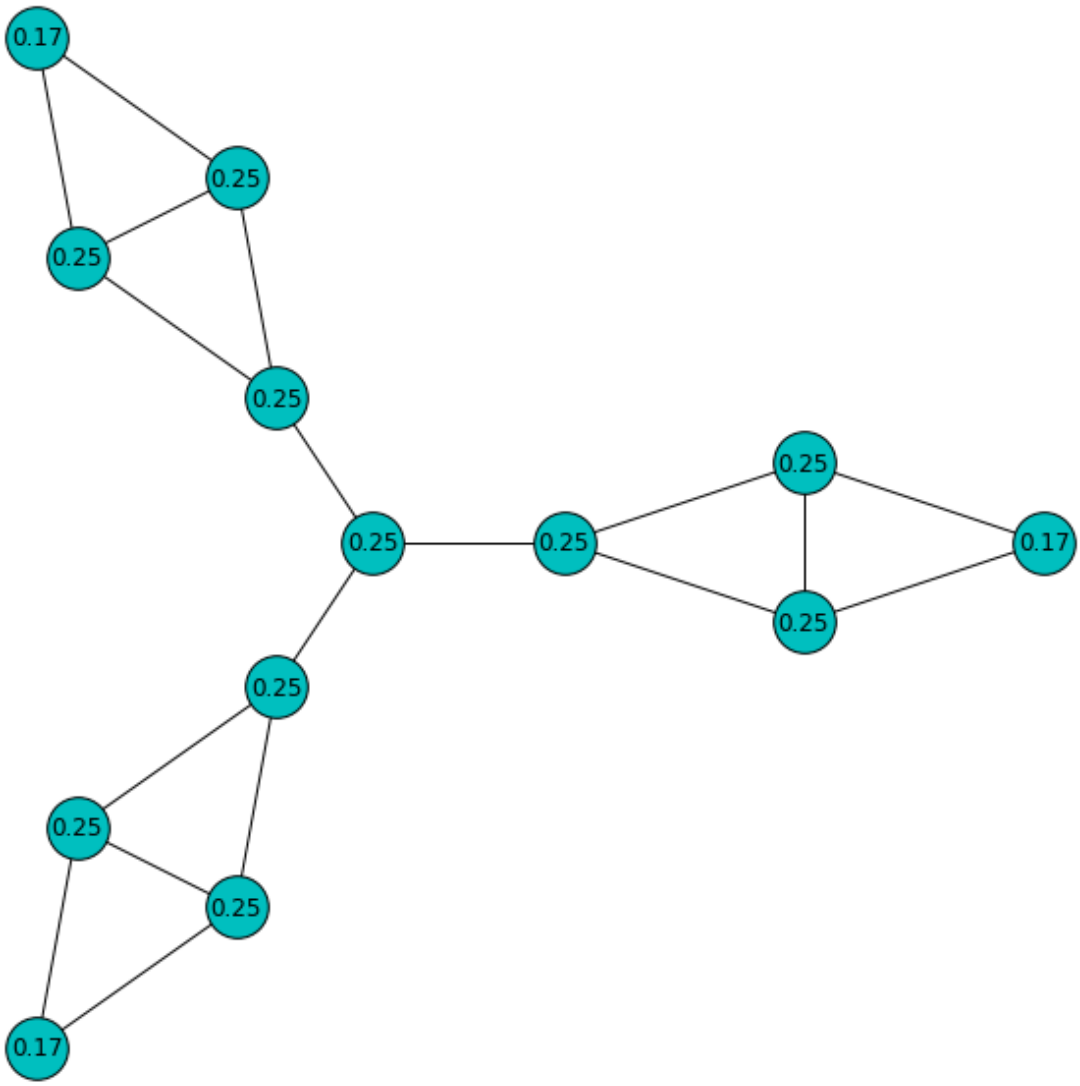


In [3]:
```
# normalized degree centrality
deg = roffdict(nx.degree_centrality(G))
plt.figure(1,figsize=(6,6))
nl.draw_atlas(G, node_color='c', node_size=1000, labels=deg)
```



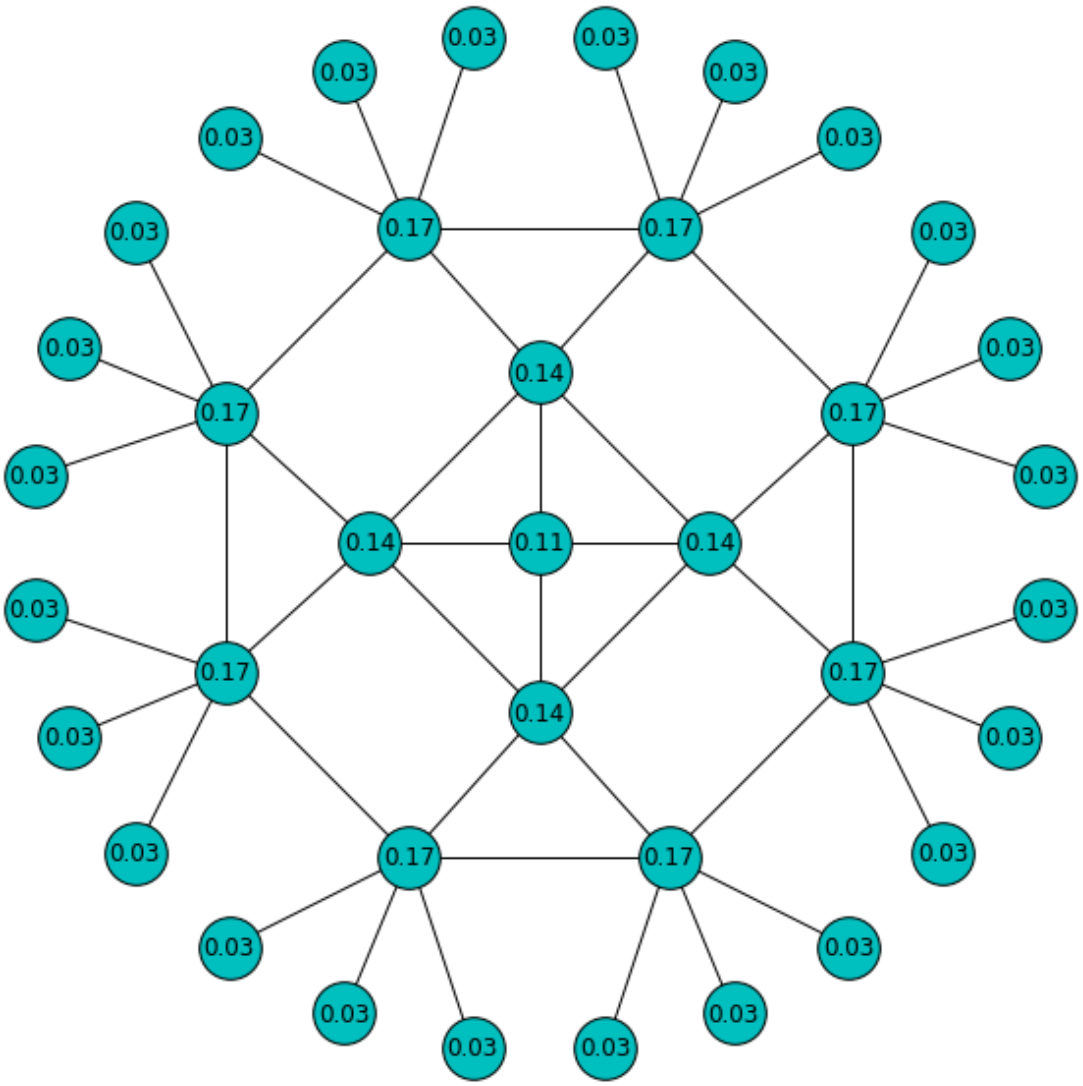
In [4]:
```
1 nl.draw_atlas??
```

In [5]:
```
1 G = nx.read_gml('Graphs/C03.gml')
2 layout = nl.absolute_layout(G, 'px', 'py')
3 deg = nx.degree_centrality(G)
4 plt.figure(1,figsize=(8,8))
5 nl.draw_atlas(G, node_color='c', node_size=1000, pos=layout, labels=roffdict
```

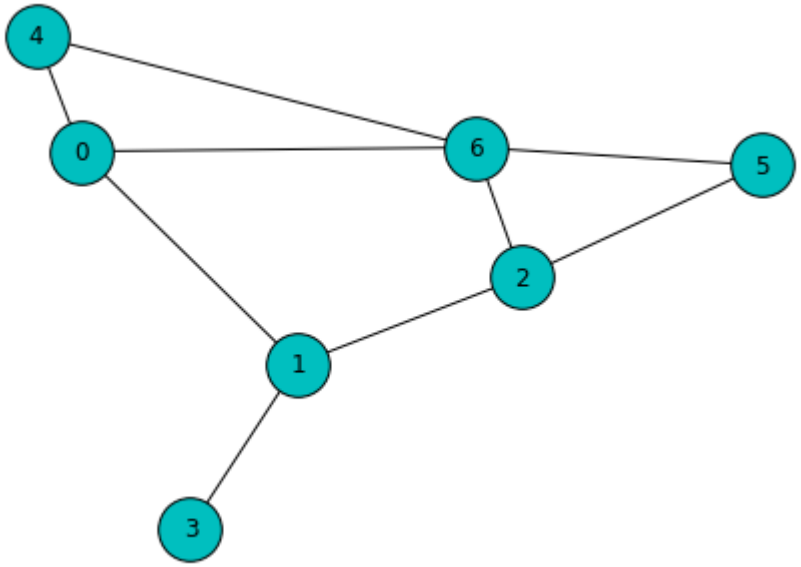


Pitfall of the degree centrality

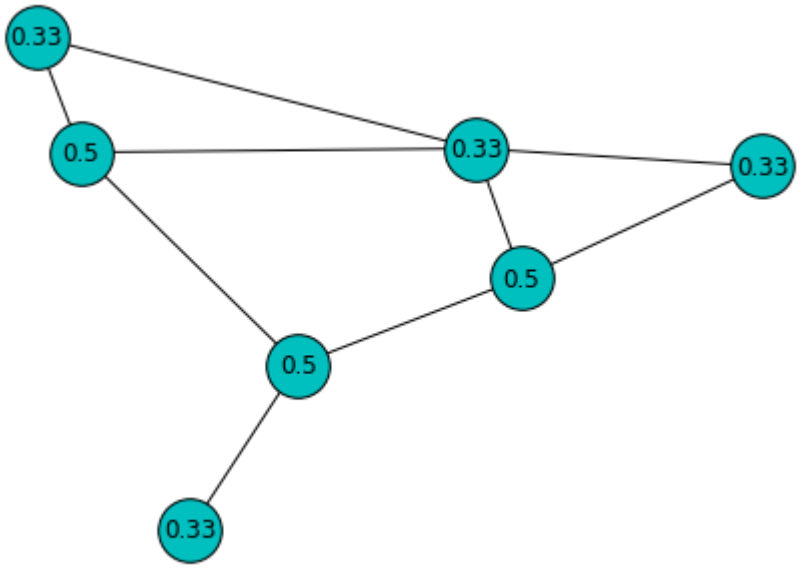
In [6]:
```
G = nx.read_gml('Graphs/C02.gml')
deg = nx.degree_centrality(G)
layout = nl.absolute_layout(G, 'px', 'py')
plt.figure(1,figsize=(8,8))
nl.draw_atlas(G, node_color='c', node_size=1000, pos=layout, labels=roffdict
```



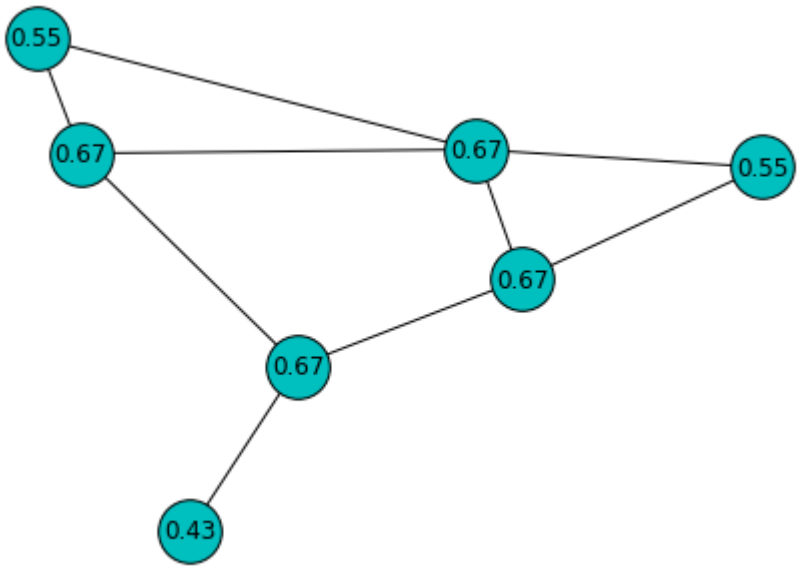
In [7]:
```
# vertex centrality
G = nx.read_gml('Graphs/g35.gml', label='id')
layout = nx.spring_layout(G)
nl.draw_atlas(G, pos=layout, node_color='c', node_size=1000)
# inverse of eccentricity
```



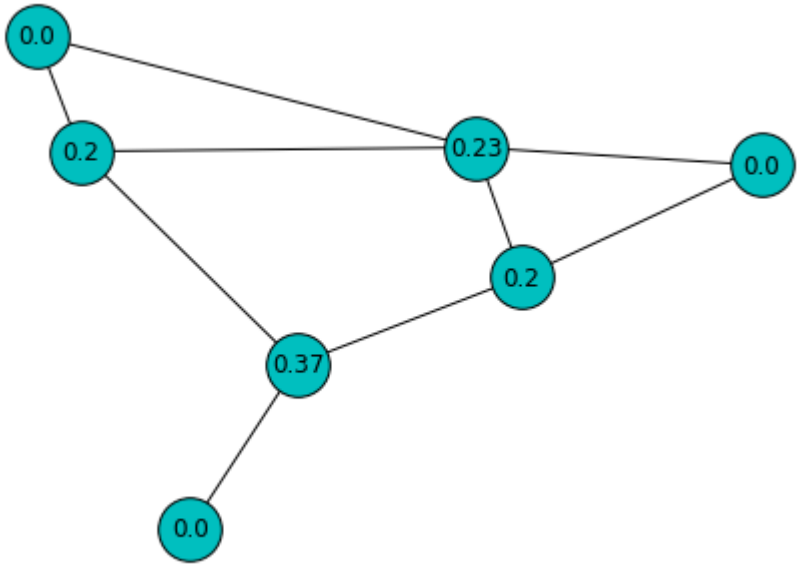
In [8]:
```
c = 1.0 / np.array(list(nx.eccentricity(G).values()))
vc = dict(zip(G.nodes(), roff(c)))
nl.draw_atlas(G, pos=layout, node_color='c', node_size=1000, labels=vc)
```



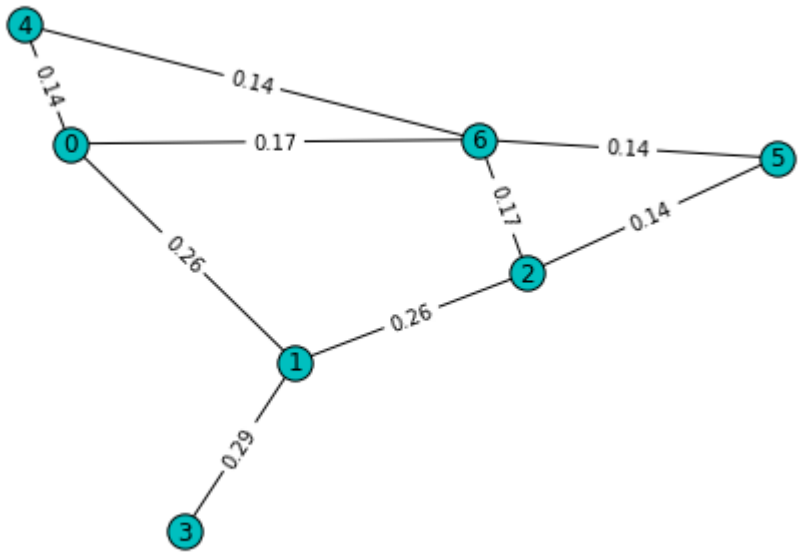
In [9]:
```
# closeness centrality
cc = nx.closeness_centrality(G)
nl.draw_atlas(G, pos=layout, node_color='c', node_size=1000, labels=roffdict
```



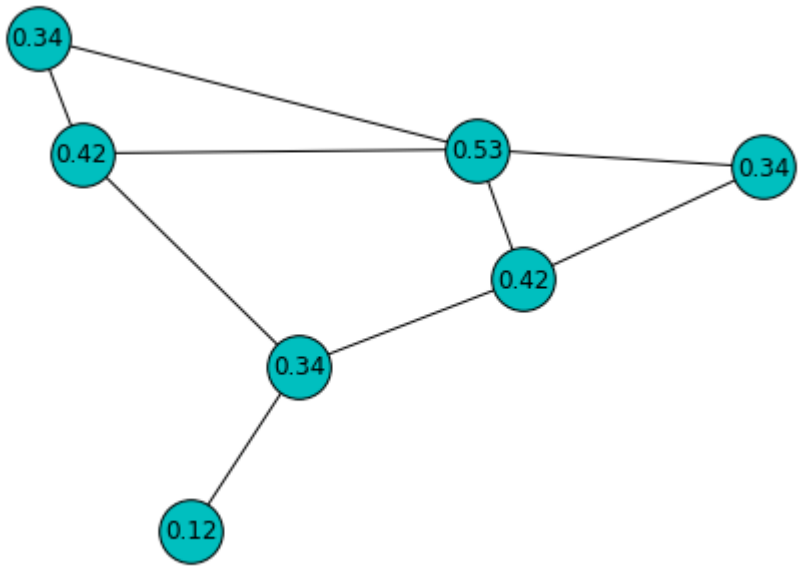
In [10]:
```
# Betweenness Centrality
bc = nx.betweenness_centrality(G)
nl.draw_atlas(G, pos=layout, node_color='c', node_size=1000, labels=roffdict
```



In [11]:
```
# nx.edge_betweenness_centrality(G)
ebc = nx.edge_betweenness_centrality(G)
nl.draw_atlas(G, pos=layout, node_color='c', edge_labels=roffdict(ebc))
```



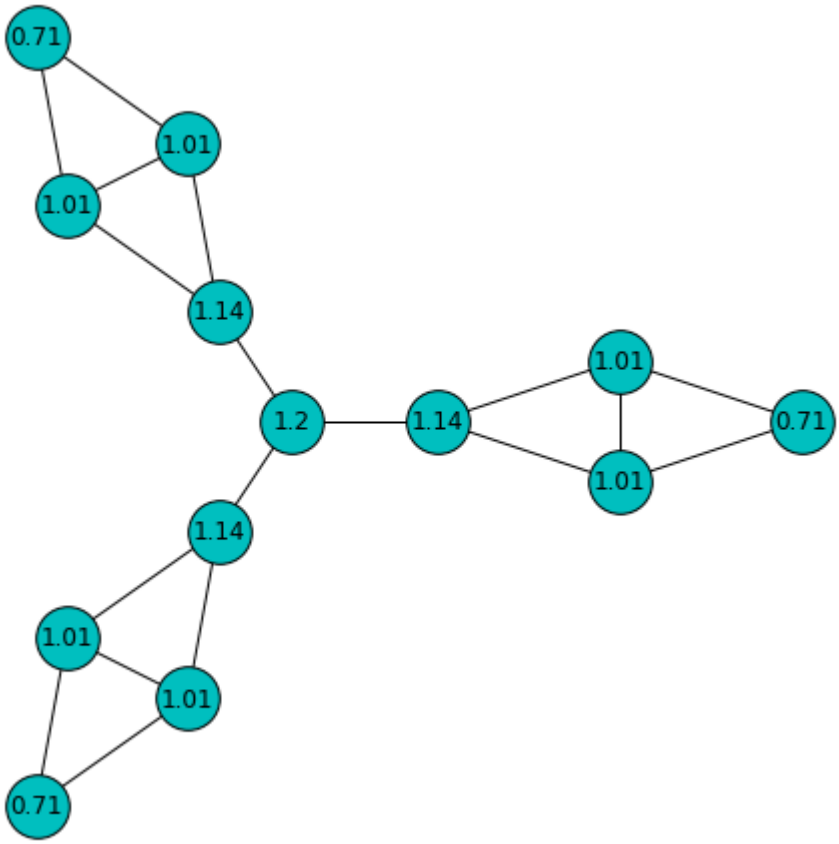
In [12]:
```
# Eigenvector Centrality
eigc = nx.eigenvector_centrality(G)
nl.draw_atlas(G, pos=layout, node_color='c', node_size=1000, labels=roffdict
```



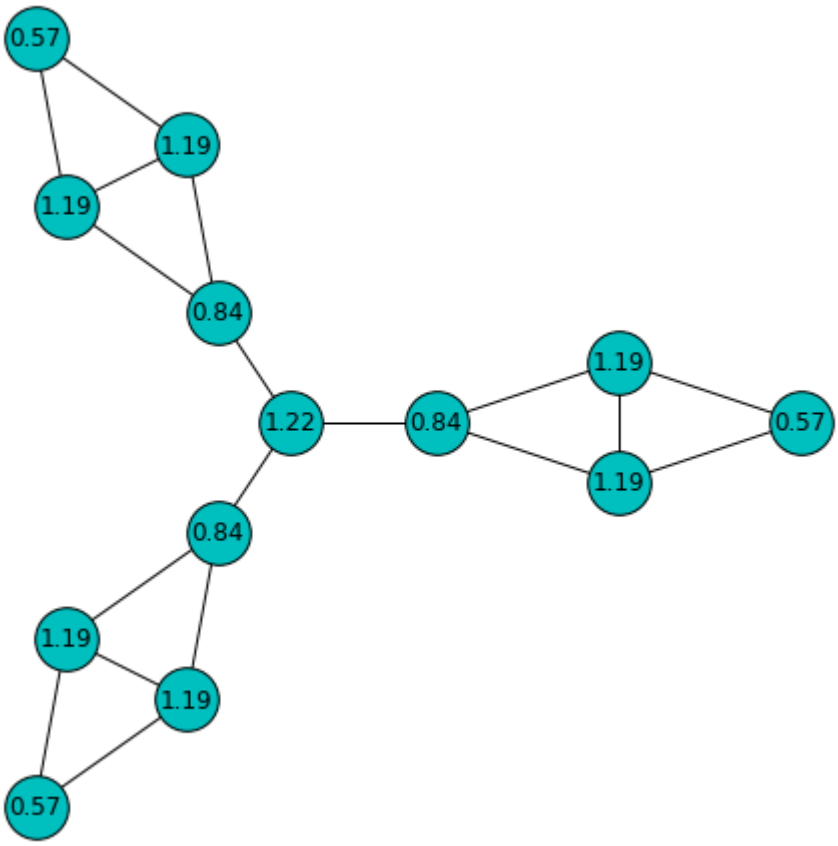
In [13]:
```
# Communicability centrality
#comc = nx.communicability_centrality(G)   # not available any more
#nl.draw_atlas(G, pos=layout, node_color='c', node_size=1000, labels=roffdic
```

In [14]:
```
def bonacich_power_centrality(G,alpha,beta):
    nodelist = G.nodes()
    R = nx.to_numpy_matrix(G, nodelist)
    I = np.identity(len(G))
    one = np.ones([len(G),1])
    B = alpha * np.linalg.inv(I - beta * R) * R
    C = B.dot(one)
    bpc = dict(zip(nodelist, map(float, C)))
    return bpc
```

In [15]:
```
G = nx.read_gml('Graphs/C03.gml')
layout = nl.absolute_layout(G, 'px', 'py')
b0 = bonacich_power_centrality(G, alpha=0.003, beta=0.35)
plt.figure(1,figsize=(6,6))
nl.draw_atlas(G, pos=layout, node_color='c', node_size=1000, labels=roffdict
```



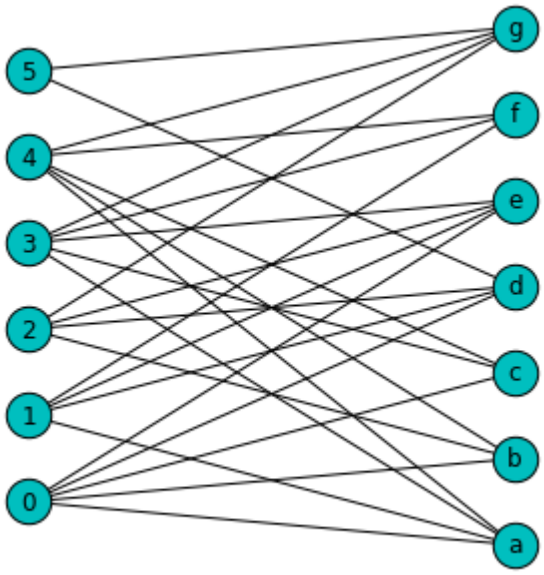
In [16]:
```
b1 = bonacich_power_centrality(G, 0.7, -0.35)
plt.figure(1,figsize=(6,6))
nl.draw_atlas(G, pos=layout, node_color='c', node_size=1000, labels=roffdict
```



Collaboration Networks

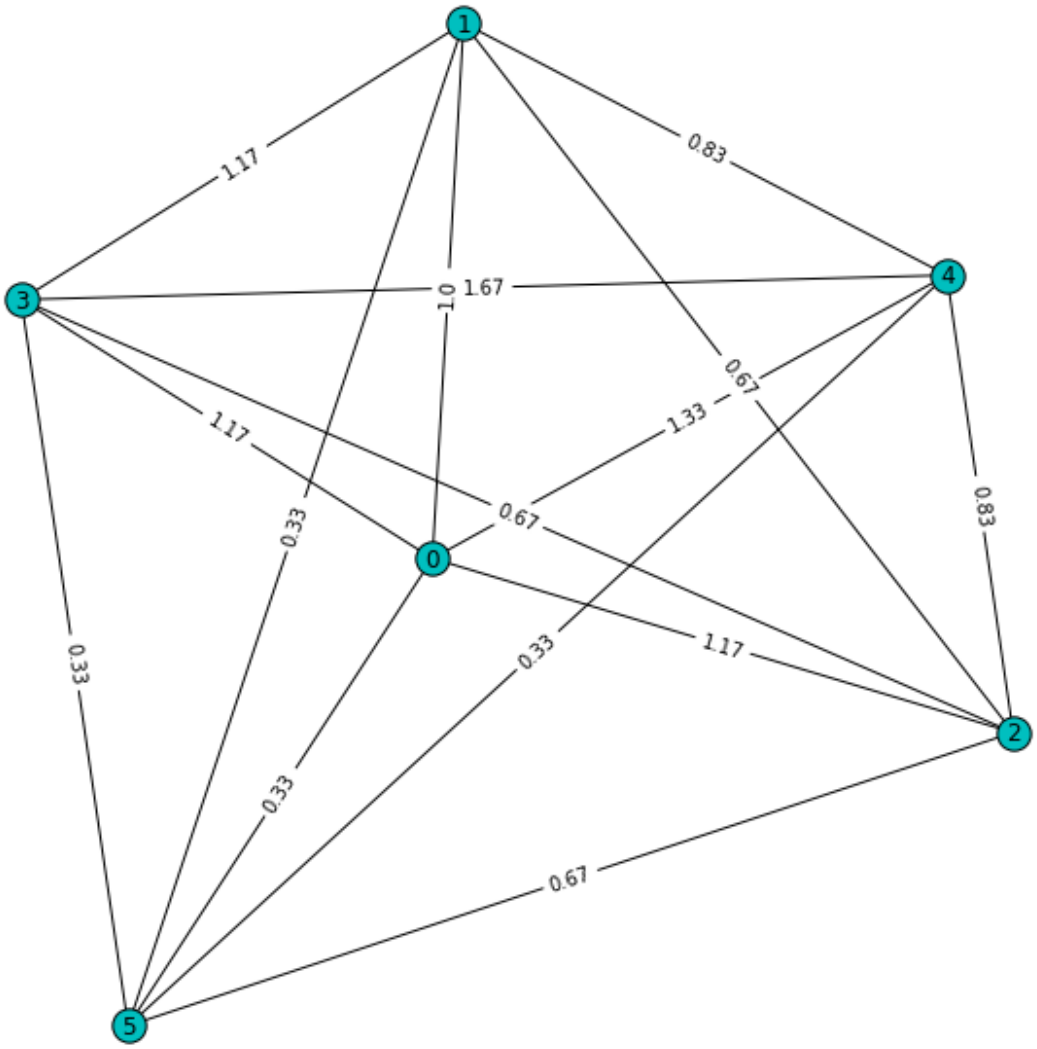
Newman's collaboration weight

In [17]:
```
B = nx.read_gml('Graphs/C04.gml')
layout=nl.absolute_layout(B, 'px', 'py')
plt.figure(1,figsize=(4,4))
nl.draw_atlas(B, node_color='c', pos=layout, node_size=500)
```



In [18]:
```
from networkx.algorithms import bipartite
bottom = list(range(6))
G = bipartite.collaboration_weighted_projected_graph(B, bottom)
layout=nx.spring_layout(G)
weight = nx.get_edge_attributes(G, 'weight')
print (roffdict(weight))
plt.figure(1,figsize=(8,8))
nl.draw_atlas(G, pos=layout, node_color='c', edge_labels=roffdict(weight))

```

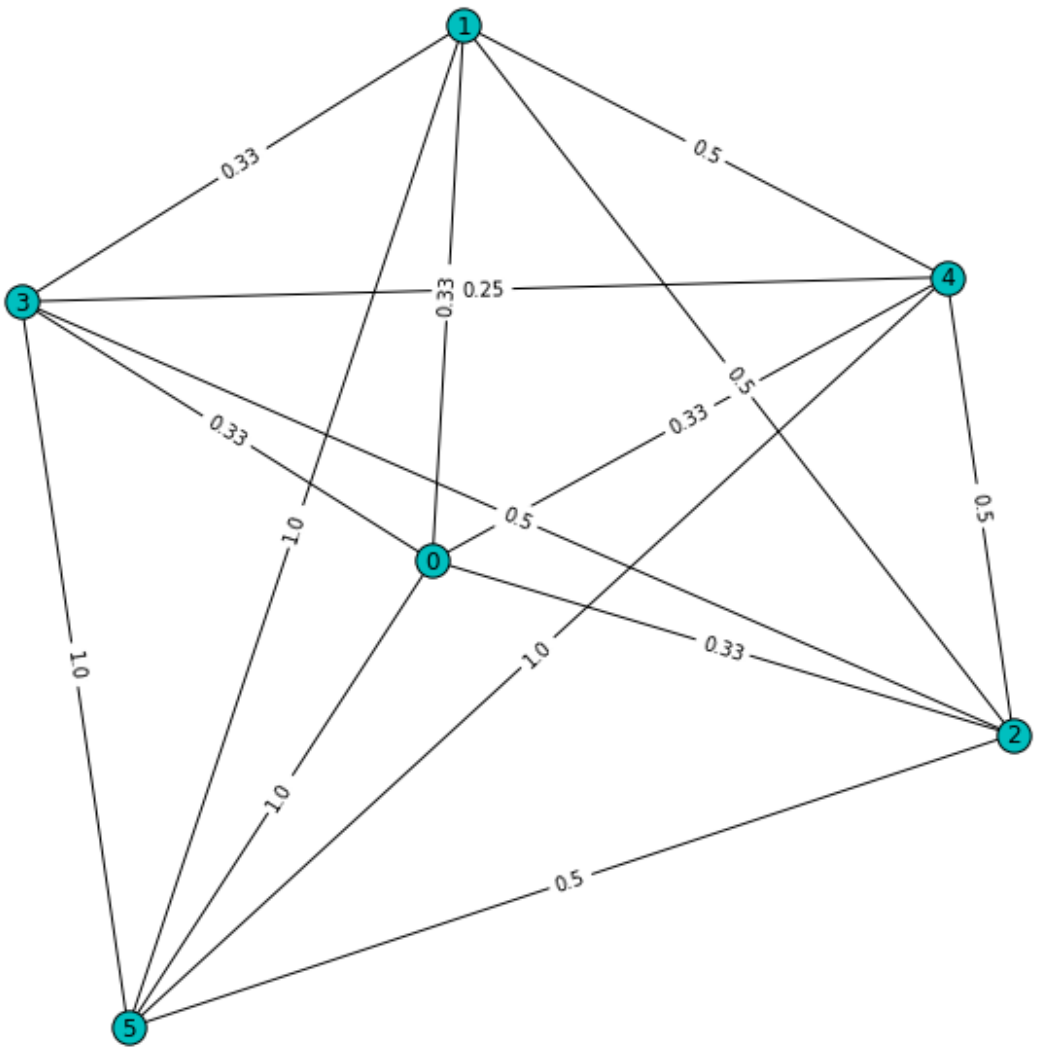
```
{(0, 1): 1.0, (0, 2): 1.17, (0, 3): 1.17, (0, 4): 1.33, (0, 5): 0.33, (1, 2):
0.67, (1, 3): 1.17, (1, 4): 0.83, (1, 5): 0.33, (2, 3): 0.67, (2, 4): 0.83, (2,
5): 0.67, (3, 4): 1.67, (3, 5): 0.33, (4, 5): 0.33}
```



bras-amoros collaboration distance

In [19]:
```
def bras_amoros_collaboration(B, bottom):
    G = nx.Graph()
    G.add_nodes_from(bottom)

    for x in bottom:
        for y in bottom:
            if x == y:
                continue
            cnt = len(set(B.neighbors(x)) & set(B.neighbors(y)))
            if cnt > 0:
                G.add_edge(x,y,weight=1.0/cnt)
    return G


```

In [20]:
```
bcd = bras_amoros_collaboration(B, bottom)
bweight = nx.get_edge_attributes(bcd, 'weight')
plt.figure(1,figsize=(8,8))
nl.draw_atlas(bcd, pos=layout, node_color='c', edge_labels=roffdict(bweight)
```



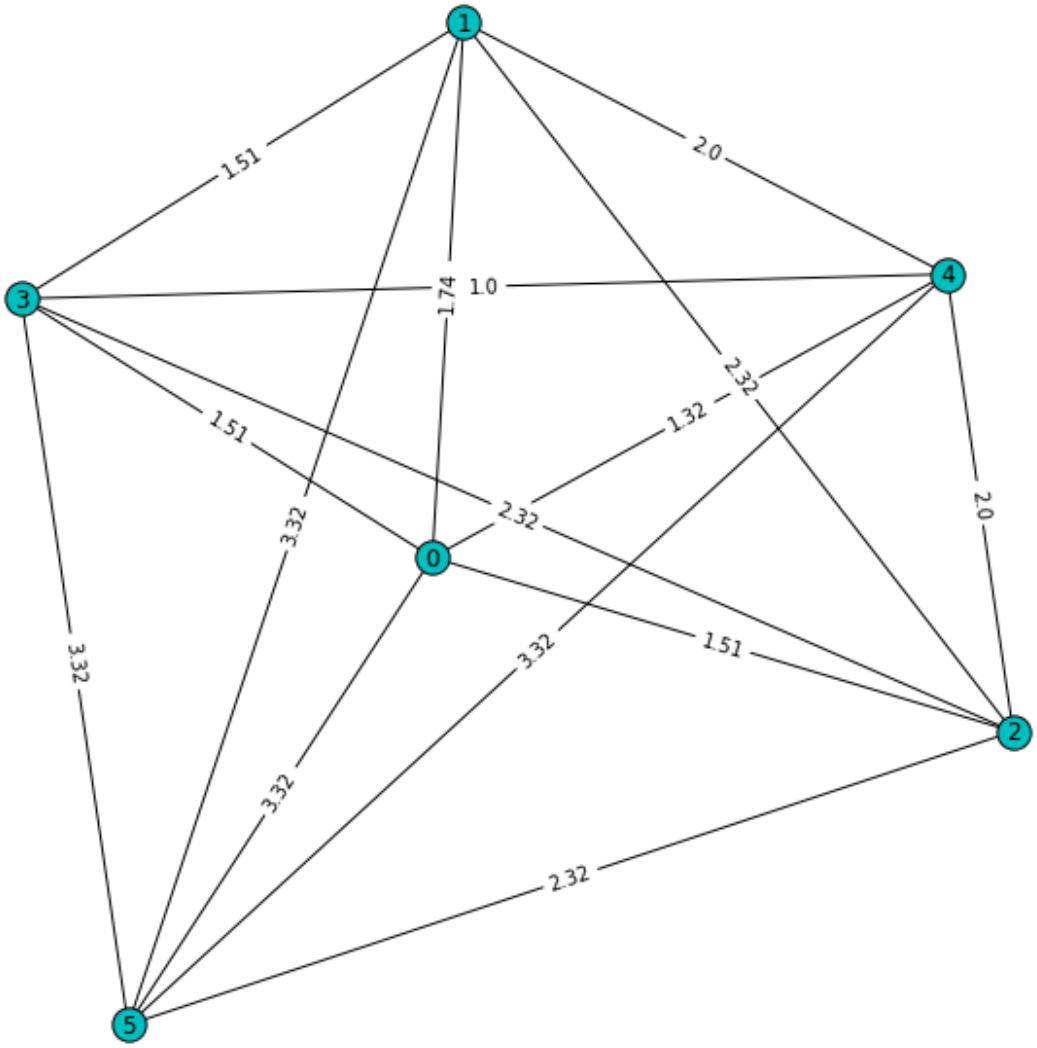
kim's collaboration distance

In [21]:

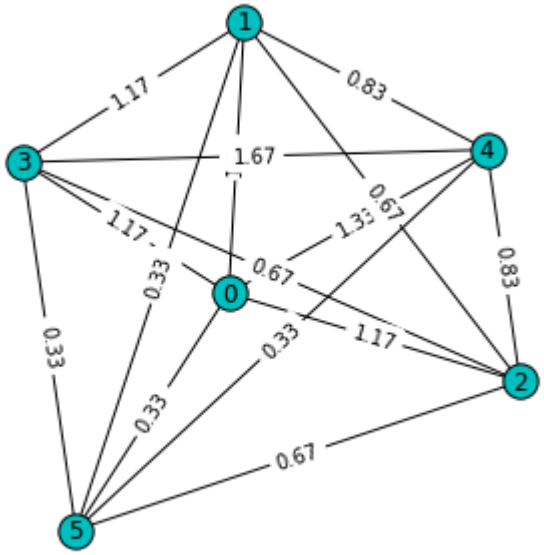
```
# kim's collaboration distance
def kim_collaboration(B, bottom):
    G = bipartite.collaboration_weighted_projected_graph(B, bottom)
    attrs = nx.get_edge_attributes(G, 'weight')
    edgelist = attrs.keys()
    weight = attrs.values()
    wmax = max(weight)
    kcd = [-np.log2(float(x)/(2*wmax)) for x in weight]
    nx.set_edge_attributes(G, values=dict(zip(edgelist,kcd)), name='kcw')
    return G
```

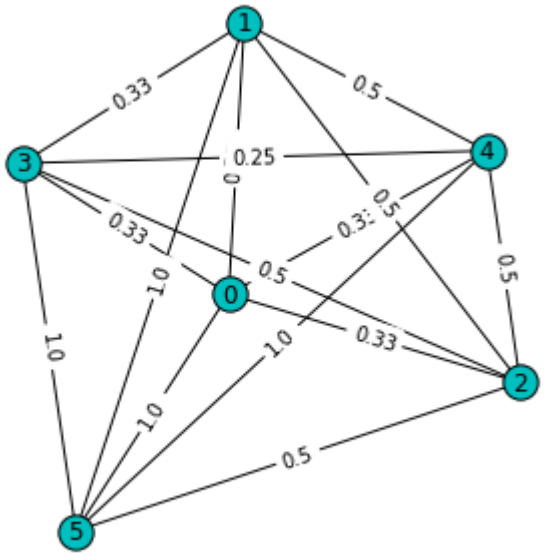
In [22]:
```
kim = kim_collaboration(B, bottom)
kcw = nx.get_edge_attributes(kim, 'kcw')
print (roffdict(kcw))
plt.figure(1,figsize=(8,8))
nl.draw_atlas(kim, pos=layout, node_color='c', edge_labels=roffdict(kcw))

```

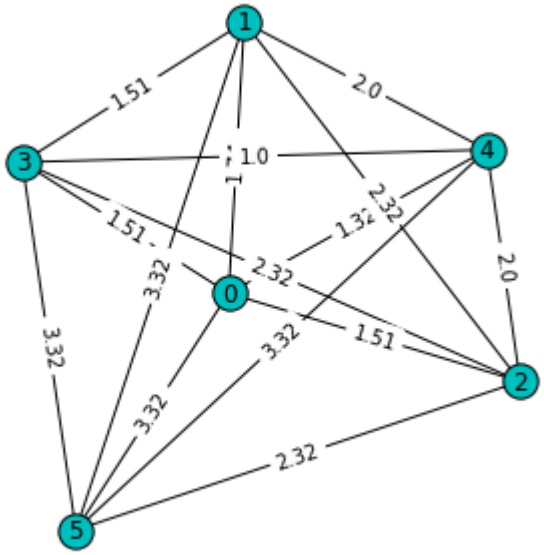
```
{(0, 1): 1.74, (0, 2): 1.51, (0, 3): 1.51, (0, 4): 1.32, (0, 5): 3.32, (1, 2):
2.32, (1, 3): 1.51, (1, 4): 2.0, (1, 5): 3.32, (2, 3): 2.32, (2, 4): 2.0, (2,
5): 2.32, (3, 4): 1.0, (3, 5): 3.32, (4, 5): 3.32}
```



In [23]:
```
plt.figure(1,figsize=(4,4))
nl.draw_atlas(G, pos=layout, node_color='c', edge_labels=roffdict(weight))
plt.figure(2,figsize=(4,4))
nl.draw_atlas(bcd, pos=layout, node_color='c', edge_labels=roffdict(bweight))
plt.figure(3,figsize=(4,4))
nl.draw_atlas(kim, pos=layout, node_color='c', edge_labels=roffdict(kcw))
```

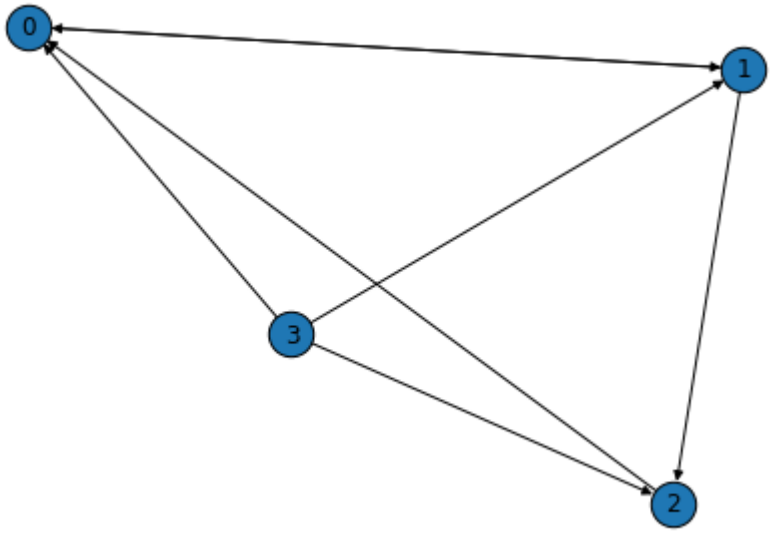






page-rank

In [24]:
```
1  G = nx.DiGraph()
2  G.add_edge(0,1)
3  G.add_edge(1,0)
4  G.add_edge(1,2)
5  G.add_edge(2,0)
6  G.add_edge(3,0)
7  G.add_edge(3,1)
8  G.add_edge(3,2)
9  nl.draw_atlas(G, node_size=500)
10 # nx.pagerank()
```



In [25]:

```
N = [1,10,11,100]
T = [0.2, 0.002, 0.001, 2e-5]
print ('%3s  %8s  %s' % ('n', 't', 'page rank'))
print ('%3s  %8s  %s' % ('---', '-------', '-----------'))
for n, t in zip(N, T):
    p = nx.pagerank(G, max_iter=n, tol=t, weight=None)
    pval = list(roff(list(p.values()),4))
    print ('%3d  %7f  %s' % (n, t, pval))
```

```
  n         t  page rank
---   -------  -----------
  1  0.200000  [0.4271, 0.3208, 0.2146, 0.0375]
 10  0.002000  [0.3814, 0.3736, 0.2075, 0.0375]
 11  0.001000  [0.3833, 0.3724, 0.2069, 0.0375]
100  0.000020  [0.3825, 0.3732, 0.2068, 0.0375]
```