Protein

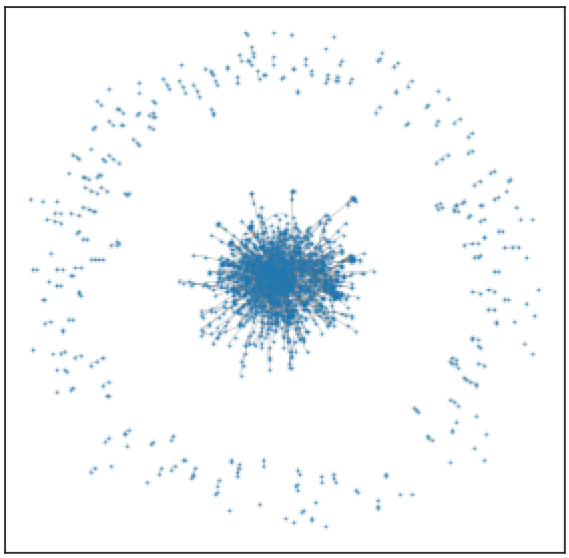
In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
G = nx.read_edgelist('networks/networks_barabasi/protein.edgelist.txt',create_using=nx.
Graph(), nodetype = int)
```

Visualização

In [2]:

```
plt.figure(figsize=(5,5))
nx.draw_networkx(G, node_size=1, edge_color='grey', alpha=0.5, width=0.5, with_labels=F
alse)
plt.show()
```



Caracterização

In [3]:

```
n = G.number_of_nodes()
m = G.number_of_edges()
print('Número de vértices:', n)
print('Número de arestas:', m)
print('Grafo conexo?',nx.is_connected(G))
```

```
Número de vértices: 2018
Número de arestas: 2930
Grafo conexo? False
```

1) Grau

In [4]:

```
degrees = np.array([val for (node, val) in G.degree()])
```

In [5]:

```
from statistics import median
print('Máximo:', degrees.max())
print('Mínimo:', degrees.min())
print('Média:',degrees.mean())
print('Mediana:', median(degrees))
print('Desvio padrão:', degrees.std())
```

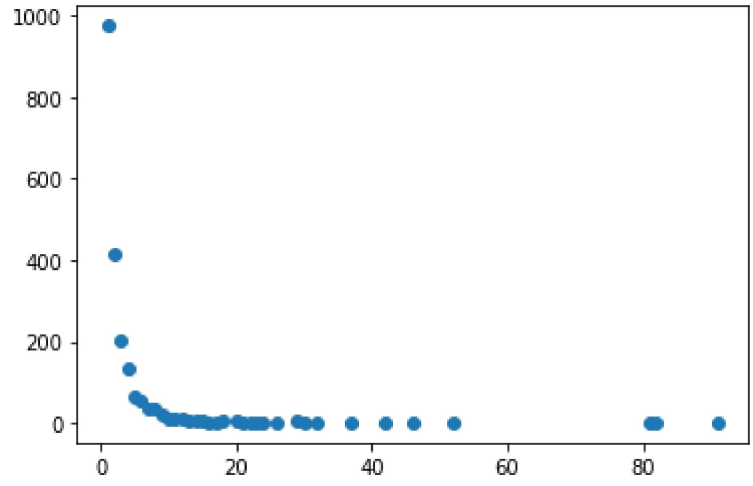
```
Máximo: 91
Mínimo: 1
Média: 2.9038652130822595
Mediana: 2.0
Desvio padrão: 4.881923426132511
```

Distribuição empírica

PMF:

In [6]:

```
#f = np.array(nx.degree_histogram(G))/n
#f = f[1:] # elimina o grau 0
x, f = np.unique(degrees, return_counts=True)
plt.scatter(x,f)
#plt.xticks(range(1,len(f)+1))
plt.show()
```



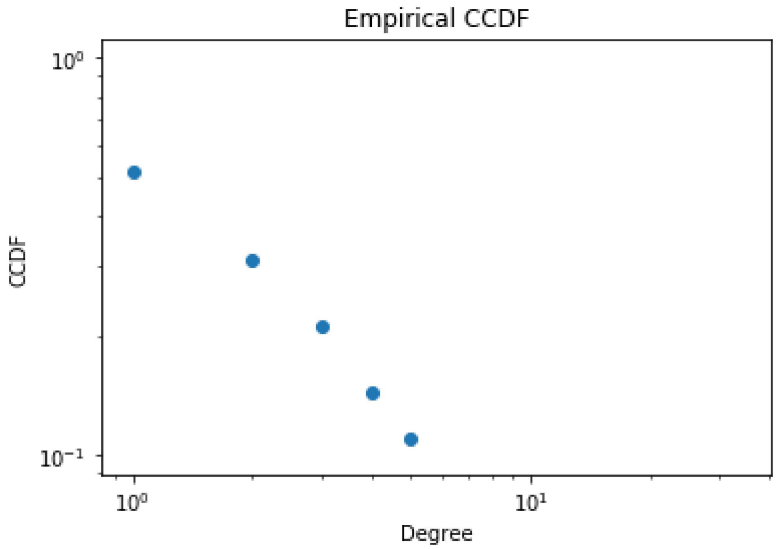
CCDF:

In [7]:

```
cdf = f.cumsum()/f.sum()
ccdf = 1-cdf

# Outro médodo:
#ccdf2 = np.zeros(len(f))
#for k,_ in enumerate(f):
#    ccdf2[k]  = 1-f[0:k+1].sum()

plt.title('Empirical CCDF')
plt.scatter(range(1,len(f)+1),ccdf)
#plt.plot(range(1,len(f)+1),ccdf2)
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Degree')
plt.ylabel('CCDF')
plt.show()
```



2) Distância

Calculando a distância entre cada par de vértices:

In [8]:

```
dist = np.array([])
for v in G.nodes():
    spl = nx.single_source_shortest_path_length(G, v) # distância short path lenght
    spl2 = dict((v2,d) for v2,d in spl.items() if v2 != v) # distância exluindo d(v,v)
    dist = np.append(dist, list(spl2.values()))
```

Estatísticas básicas:

In [9]:

```
print('Máximo:', dist.max())
print('Mínimo:', dist.min())
print('Média:',dist.mean())
print('Mediana:', median(dist))
print('Desvio padrão:', dist.std())
```

```
Máximo: 14.0
Mínimo: 1.0
Média: 5.6109290866980395
Mediana: 6.0
Desvio padrão: 1.646376867828377
```

Verificando o cálculo da distância média através do método disponibilizado pela biblioteca:

In [41]:

```
if nx.is_connected(G):
    d_mean = nx.average_shortest_path_length(G)
    print('Distância média:',d_mean)
else:
    print('Rede não é conexa!')
```

```
Rede não é conexa!
```

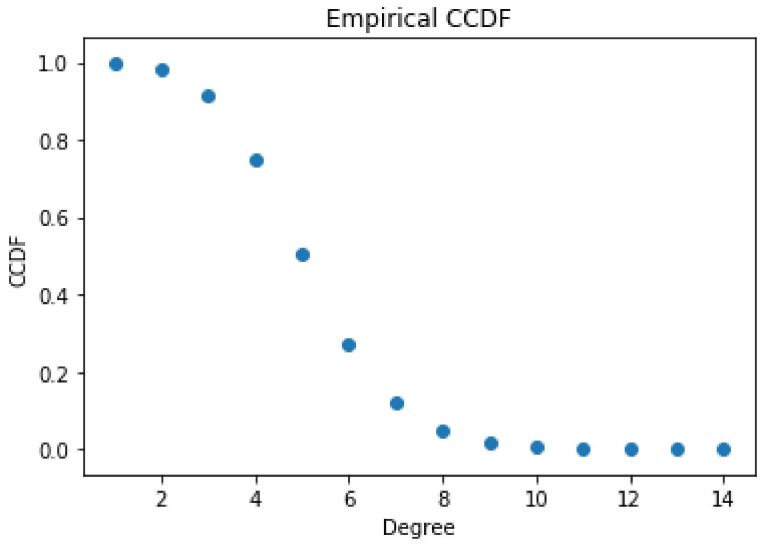
Frequência relativa e CCDF:

In [42]:

```
from scipy.special import comb
Lmax = comb(n,2) # número máximo de arestas

counts, f_d = np.unique(dist, return_counts=True)
cdf = f_d.cumsum()/f_d.sum()
ccdf = 1-cdf

plt.scatter(counts,ccdf)
plt.title('Empirical CCDF')
#plt.xscale('log')
#plt.yscale('log')
plt.xlabel('Degree')
plt.ylabel('CCDF')
plt.show()
```



3) Tamanho das componentes conexas

Número de componentes conexas:

In [43]:

```
nx.number_connected_components(G)
```

Out[43]:

185

Tamanho das componentes:

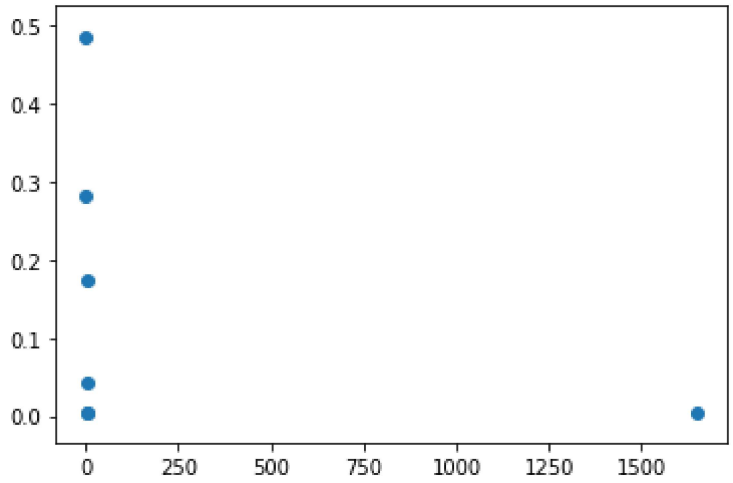
In [40]:

```
len_cc = np.array([len(c) for c in sorted(nx.connected_components(G), key=len, reverse=
True)])
print('Máximo:', len_cc.max())
print('Mínimo:', len_cc.min())
print('Média:',len_cc.mean())
print('Mediana:', median(len_cc))
print('Desvio padrão:', len_cc.std())
```

```
Máximo: 1647
Mínimo: 1
Média: 10.908108108108108
Mediana: 2
Desvio padrão: 120.61745285927152
```

In [47]:

```
x, f = np.unique(len_cc, return_counts=True)
pmf = f/f.sum()
plt.scatter(x,pmf)
plt.show()
```



In [46]:

```
len_cc
```

Out[46]:

```
array([1647,    6,    5,    4,    4,    4,    4,    4,    4,    4,    4,
          3,    3,    3,    3,    3,    3,    3,    3,    3,    3,    3,
          3,    3,    3,    3,    3,    3,    3,    3,    3,    3,    3,
          3,    3,    3,    3,    3,    3,    3,    3,    3,    3,    2,
          2,    2,    2,    2,    2,    2,    2,    2,    2,    2,    2,
          2,    2,    2,    2,    2,    2,    2,    2,    2,    2,    2,
          2,    2,    2,    2,    2,    2,    2,    2,    2,    2,    2,
          2,    2,    2,    2,    2,    2,    2,    2,    2,    2,    2,
          2,    2,    2,    2,    2,    2,    2,    2,    2,    2,    2,
          2,    2,    2,    2,    2,    2,    2,    2,    2,    2,    2,
          2,    2,    2,    2,    2,    2,    2,    2,    2,    2,    2,
          2,    1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
          1,    1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
          1,    1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
          1,    1,    1,    1,    1,    1,    1,    1,    1,    1,    1,
          1,    1,    1,    1,    1,    1,    1,    1,    1])
```

4) Clusterização

4.1) Clusterização Local

In [13]:

```
cluster = np.array(list(nx.clustering(G).values()))
print('Máximo:', cluster.max())
print('Mínimo:', cluster.min())
print('Média:', cluster.mean())
print('Mediana:', median(cluster))
print('Desvio padrão:', cluster.std())
```

```
Máximo: 1.0
Mínimo: 0.0
Média: 0.046194001297365124
Mediana: 0.0
Desvio padrão: 0.17603114068332915
```

Verificando a clusterização média:

In [14]:

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

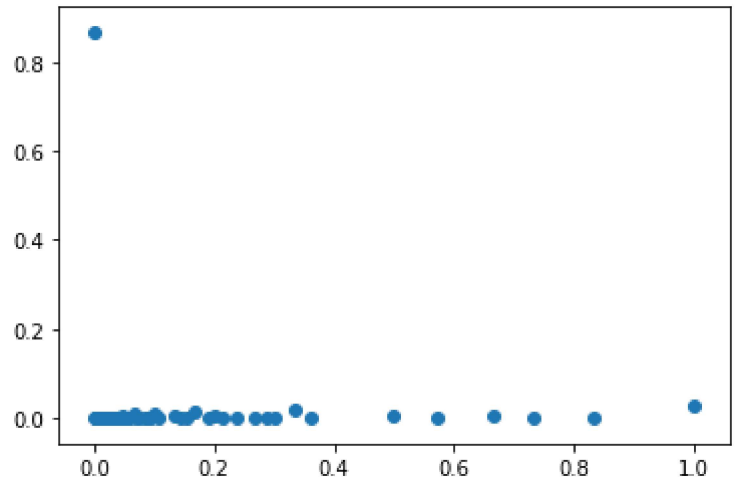
Out[14]:

```
0.046194001297365166
```

PMF:

In [15]:

```
x, f = np.unique(cluster, return_counts=True)
pmf = f/f.sum()
plt.scatter(x,pmf)
plt.show()
```

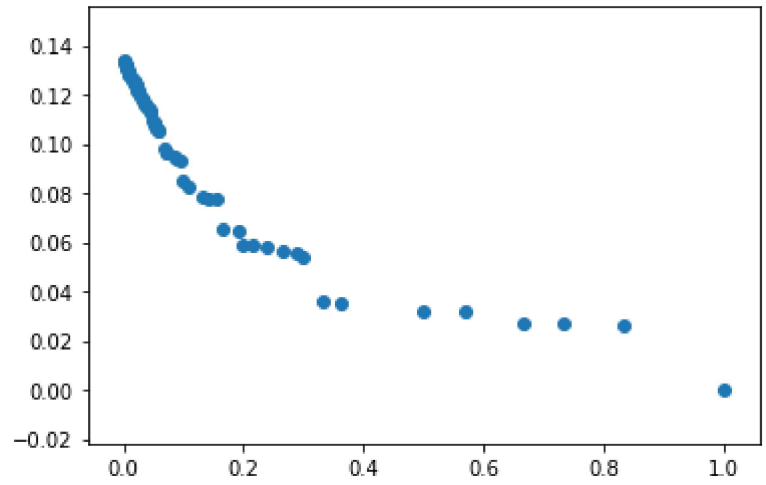


CCDF:

In [16]:

```
x, f = np.unique(cluster, return_counts=True)
cdf = f.cumsum()/f.sum()
ccdf = 1-cdf

plt.scatter(x,ccdf)
plt.show()
```



4.2) Clusterização global

In [17]:

```
n_triang = np.array(list(nx.triangles(G).values())).sum()/3 # método conta 3 vezes o tr
iângula (1x para cada vértice)
print('Número de triângulos:', n_triang)
```

```
Número de triângulos: 212.0
```

Clusterização global:

In [18]:

```
nx.transitivity(G)
```

Out[18]:

```
0.02361415364051535
```

5) Centralidade

5.1) Centralidade de Grau

In [19]:

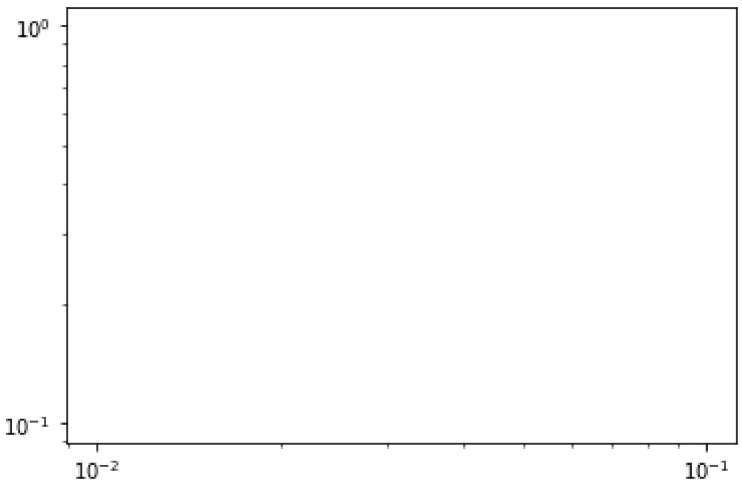
```
cent = np.array(list(nx.degree_centrality(G).values()))
print('Máximo:', cent.max())
print('Mínimo:', cent.min())
print('Média:', cent.mean())
print('Mediana:', median(cent))
print('Desvio padrão:', cent.std())
```

```
Máximo: 0.0451165096678235
Mínimo: 0.0004957858205255329
Média: 0.0014396951973635397
Mediana: 0.0009915716410510659
Desvio padrão: 0.0024203884115679276
```

CCDF:

In [20]:

```
x, f = np.unique(cent, return_counts=True)
cdf = f.cumsum()/f.sum()
ccdf = 1-cdf
plt.scatter(x,ccdf)
plt.xscale('log')
plt.yscale('log')
plt.show()
```



5.2) Betweeness

In [21]:

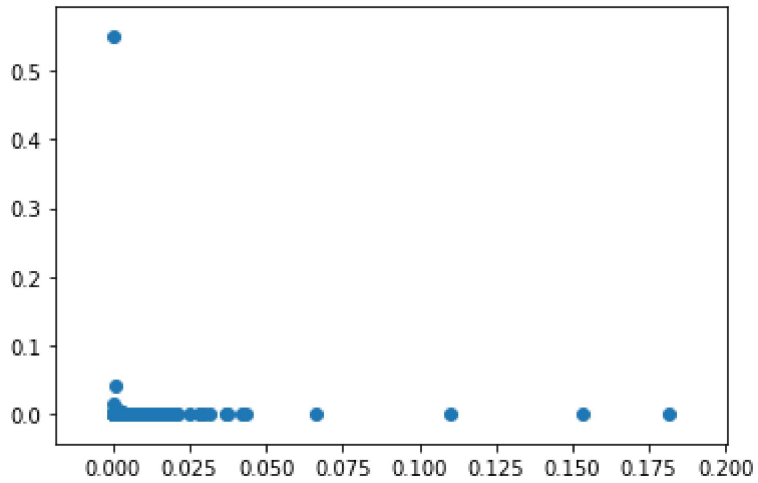
```
btw = np.array(list(nx.betweenness_centrality(G).values()))
print('Máximo:', btw.max())
print('Mínimo:', btw.min())
print('Média:', btw.mean())
print('Mediana:', median(btw))
print('Desvio padrão:', btw.std())
```

```
Máximo: 0.18151976215555363
Mínimo: 0.0
Média: 0.0015236220900641463
Mediana: 0.0
Desvio padrão: 0.006881556812597941
```

PMF:

In [22]:

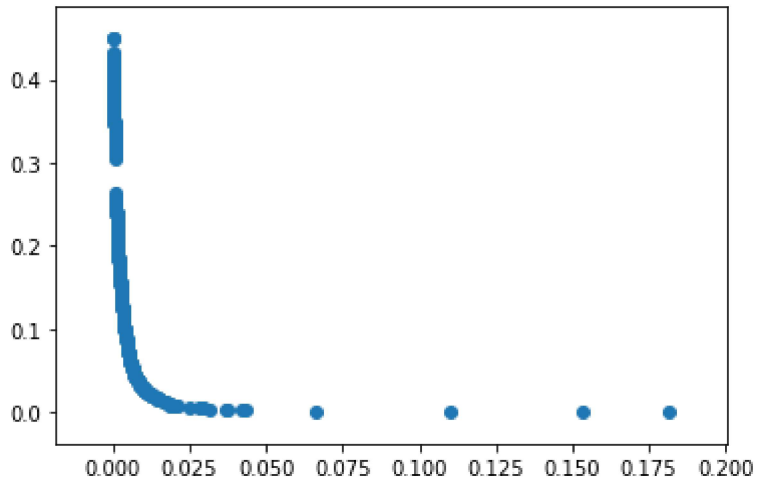
```
x, f = np.unique(btw, return_counts=True)
pmf = f/f.sum()
plt.scatter(x,pmf)
plt.show()
```



CCDF:

In [23]:

```
x, f = np.unique(btw, return_counts=True)
cdf = f.cumsum()/f.sum()
ccdf = 1-cdf
plt.scatter(x,ccdf)
plt.show()
```



Visualziação do Betweenes:

In [24]:

```
#nx.draw_circular(G, node_color=btw, cmap=plt.cm.Blues, with_labels=True)
```

5.3) Closeness

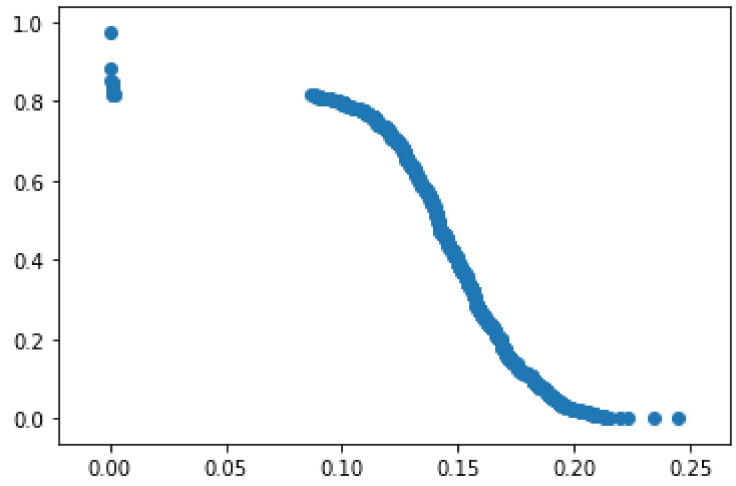
In [25]:

```
close = np.array(list(nx.closeness_centrality(G).values()))
print('Máximo:', close.max())
print('Mínimo:', close.min())
print('Média:', close.mean())
print('Mediana:', median(close))
print('Desvio padrão:', close.std())
```

```
Máximo: 0.24498275690734178
Mínimo: 0.0
Média: 0.1224292730964651
Mediana: 0.14145329150410224
Desvio padrão: 0.06225392075786067
```

In [26]:

```
x, f = np.unique(close, return_counts=True)
cdf = f.cumsum()/f.sum()
ccdf = 1-cdf
plt.scatter(x,ccdf)
plt.show()
```



In [27]:

```
#nx.draw_circular(G, node_color=close, cmap=plt.cm.Blues, with_labels=True)
```

5.4) Auto-Vetor

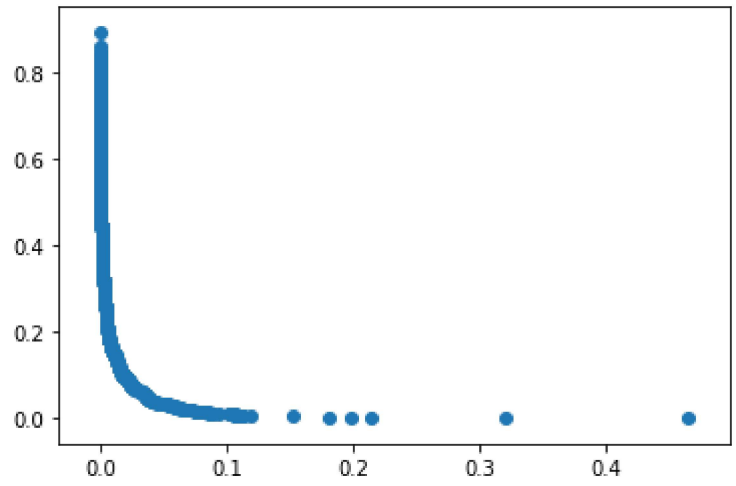
In [28]:

```
ev = np.array(list(nx.eigenvector_centrality(G).values()))
print('Máximo:', ev.max())
print('Mínimo:', ev.min())
print('Média:', ev.mean())
print('Mediana:', median(ev))
print('Desvio padrão:', ev.std())
```

```
Máximo: 0.4654668822552674
Mínimo: 4.323660793748502e-25
Média: 0.006895226561824924
Mediana: 0.0005193122373762252
Desvio padrão: 0.021165915747076533
```

In [29]:

```
x, f = np.unique(ev, return_counts=True)
cdf = f.cumsum()/f.sum()
ccdf = 1-cdf
plt.scatter(x,ccdf)
plt.show()
```



In [30]:

```
#nx.draw_circular(G, node_color=ev, cmap=plt.cm.Blues, with_labels=True)
```

5.6) PageRank

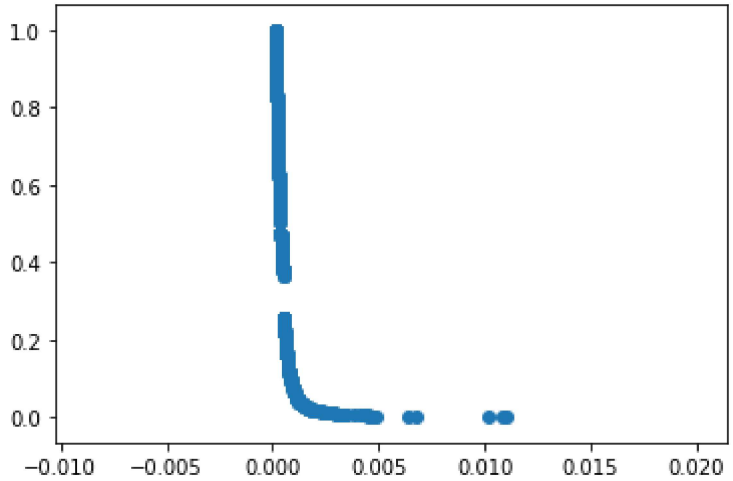
In [31]:

```
pr = np.array(list(nx.pagerank(G, alpha=0.9).values()))
print('Máximo:', pr.max())
print('Mínimo:', pr.min())
print('Média:', pr.mean())
print('Mediana:', median(pr))
print('Desvio padrão:', pr.std())
```

```
Máximo: 0.011017112062038982
Mínimo: 0.00015398258328035226
Média: 0.0004955401387512387
Mediana: 0.000381439616484525
Desvio padrão: 0.0006186174251784557
```

In [32]:

```
x, f = np.unique(pr, return_counts=True)
cdf = f.cumsum()/f.sum()
ccdf = 1-cdf
plt.scatter(x,ccdf)
#plt.xscale('log')
#plt.yscale('log')
plt.show()
```



In [33]:

```
#nx.draw_circular(G, node_color=pr, cmap=plt.cm.Blues, with_labels=True)
```

5) Similaridade

5.1) Jaccard

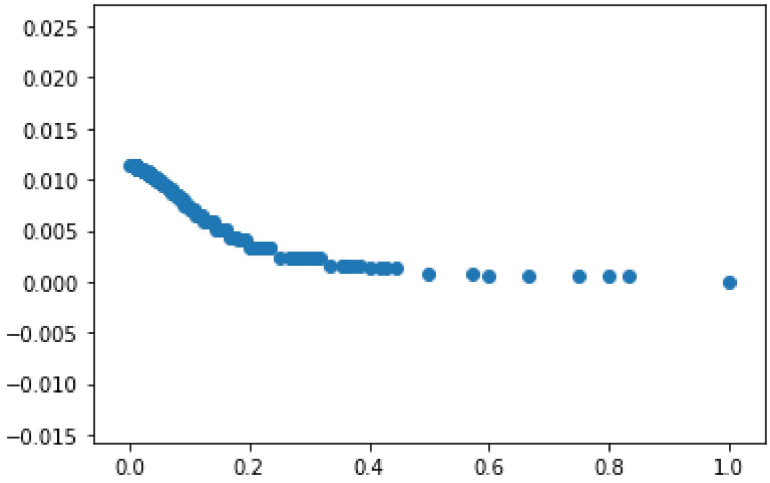
In [34]:

```
jaccard = np.array([p for (u, v, p) in nx.jaccard_coefficient(G)])
print('Máximo:', jaccard.max())
print('Mínimo:', jaccard.min())
print('Média:', jaccard.mean())
print('Mediana:', median(jaccard))
print('Desvio padrão:', jaccard.std())
```

```
Máximo: 1.0
Mínimo: 0.0
Média: 0.0024313333912945635
Mediana: 0.0
Desvio padrão: 0.033385140219770904
```

In [35]:

```
x, f = np.unique(jaccard, return_counts=True)
cdf = f.cumsum()/f.sum()
ccdf = 1-cdf
plt.scatter(x,ccdf)
#plt.xscale('log')
#plt.yscale('log')
plt.show()
```

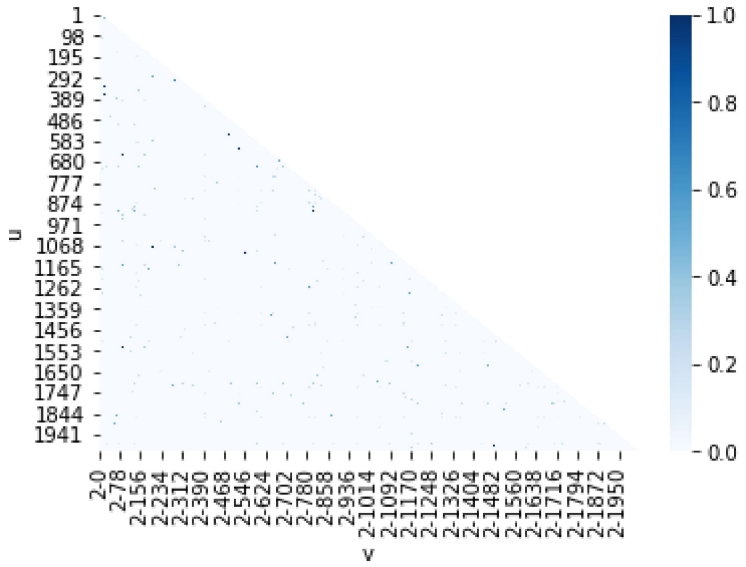


In [36]:

```
mapa = [[u,v, p] for (u, v, p) in nx.jaccard_coefficient(G)]

import pandas as pd
df = pd.DataFrame(mapa)
df.set_index([0,1],inplace=True)
df_m = df.unstack(level=0)

import seaborn as sns
ax = sns.heatmap(df_m, cmap=plt.cm.Blues)
locs, labels = plt.xticks()
#plt.xticks(locs,range(1,33,2))
plt.xlabel('v')
plt.ylabel('u')
plt.show()
```



5.2) Adamic/Adar

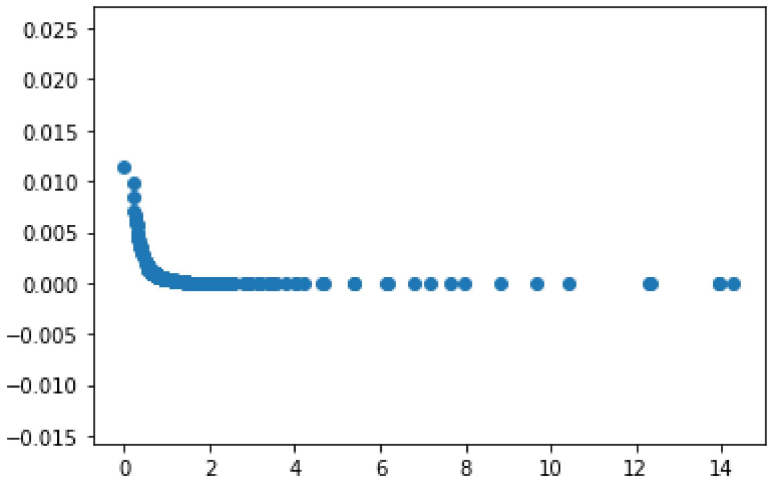
In [37]:

```
adamic = np.array([p for (u, v, p) in nx.adamic_adar_index(G)])
print('Máximo:', adamic.max())
print('Mínimo:', adamic.min())
print('Média:', adamic.mean())
print('Mediana:', median(adamic))
print('Desvio padrão:', adamic.std())
```

```
Máximo: 14.30304051308118
Mínimo: 0.0
Média: 0.004368381235791592
Mediana: 0.0
Desvio padrão: 0.05574718405356366
```

In [38]:

```
x, f = np.unique(adamic, return_counts=True)
cdf = f.cumsum()/f.sum()
ccdf = 1-cdf
plt.scatter(x,ccdf)
plt.show()
```



In [39]:

```
#pairs = dict([((u,v), p) for (u, v, p) in nx.adamic_adar_index(G)])
mapa = [[u,v, p] for (u, v, p) in nx.adamic_adar_index(G)]

import pandas as pd
df = pd.DataFrame(mapa)
df.set_index([0,1],inplace=True)
df_m = df.unstack(level=0)

import seaborn as sns
ax = sns.heatmap(df_m, cmap=plt.cm.Blues)
locs, labels = plt.xticks()
#plt.xticks(locs,range(1,34,2))
plt.xlabel('v')
plt.ylabel('u')
plt.show()
```