CSCI - 5352 (Network Analysis and Modeling) - Problem Set 6

Name of Student : Rajarshi Basak

November 28, 2018

[1] (a) (i) ***Plot for ccdf using Price's Model of a citation network :*** For choices of $c = 3$ and $n = 10^6$, shown below is a single figure showing the four complementary cumulative distribution functions $\Pr(K \geq k_{in})$ (the ccdf) for network in-degree k_{in}, one for each choice of r = 1,2,3,4.

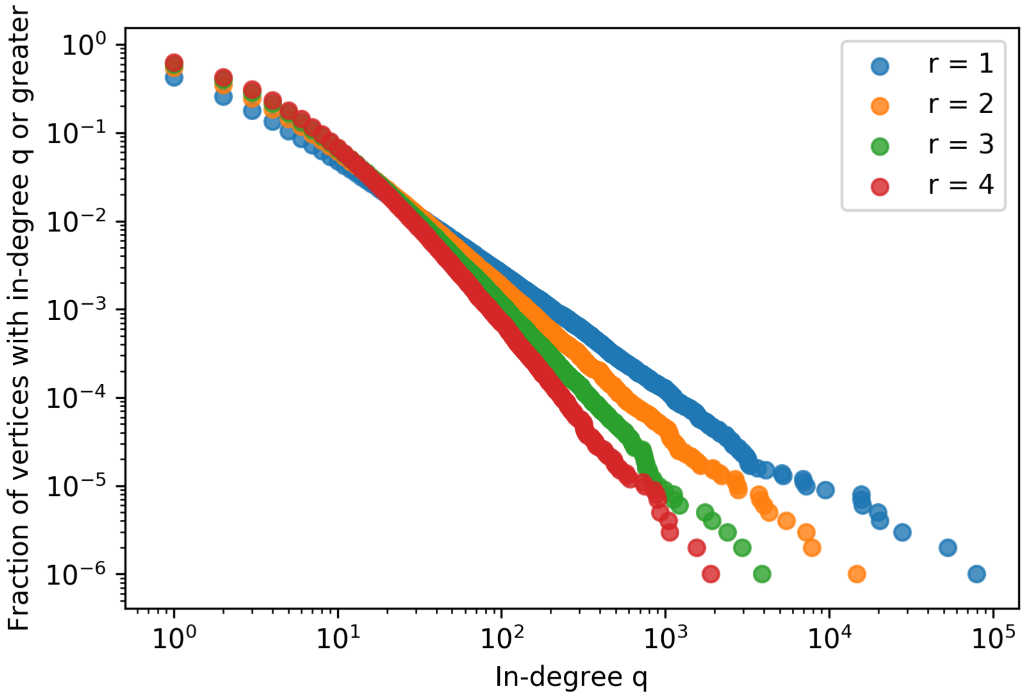

Figure 1: Plot of the complementary cumulative distribution functions versus the network in-degree for r = 1,2,3,4 using Price's Model of a citation network

[1] (a) (ii) ***Discussion :*** The shape of the curve is a straight line for a majority of the range of the in-degree q and the fraction of vertices with in-degree q or greater, when both the axes are shown on a log scale in the plot. This means that the fraction of the vertices with in-degree q or greater scales as a power law function of the in-degree q, with the numerical value of the exponent of the power law given by the slope of the graph. Since the slope is negative, the exponent is a negative number.

In general, there is a very large fraction of vertices with in-degree q or greater for a very small value of the in-degree q (fraction close to 1 for in-degree close to 1), and a really small fraction for a really large in-degree (fraction close to 10^{-6} for in-degree close to 10^5). These extreme values are linked by a linear chain of points on a logarithmic scale.

Contribution of uniform attachment mechanism on the shape of the distribution : For $r = 1$, the slope of the curve is higher (negative value closer to 0) than for $r = 2$, and the trend follows for higher values of r. Thus the greater the value of r, the lesser the value of the exponent. In other words, for low values (e.g. $r = 1$) of the uniform attachment contribution, there are a large fraction of vertices with in-degree q or greater for small q, smaller fractions for higher q, and a tiny fraction for extremely large values of q. As the value of r increases, the fraction of vertices with in-degree q or greater goes down for higher values of q. This means that we observe the 'rich get richer' effect the most when r is the smallest, and the effect diminishes as we increase the value of r.

Fraction of vertices with $k_{in} = 0$ ***:*** As mentioned before, there are a large fraction of vertices with in-degree q or greater for a small value of in-degree q. This implies that are a large fraction of vertices with in-degree q or greater when $q = 0$, which tallies with the 'rich get richer' effect. A huge number of vertices have a 0 or near-0 in-degree, and lesser and lesser vertices have a greater in-degree, and only a handful have a gargantuan in-degree.

[1] (b) (i) Using reasonable values of model parameters for real citation networks ($c = 12$ and $r = 5$), from the numerical simulation (averaged over 10 repetitions of the simulation, since the values were observed to not not fluctuate by a large amount,i.e. the variations were only in the third decimal place and beyond),

- The average number of citations to a paper (in-degree) in the first 10 % of published papers (vertices) is $\boxed{88.299}$.
- The average number of citations to a paper (in-degree) in the last 10 % of published papers (vertices) is $\boxed{0.187}$.

[1] (b) (ii) ***Discussion :*** Here we observe the 'first-mover advantage'. In the 'first mover advantage', *'the longer a vertex is in the network, the more chances it has to accumulate additional edges, and further, older vertices tend to have a larger share of citations and thus they gain additional edges faster'.*

Since on an average the first 10 % of vertices have a higher in-degree (about 81) than that for the last 10 % of vertices (about 0.18), this model shows the 'first mover advantage'. Applied to citation networks, the first papers published in a field have a greater number of citations than those published later in the same field.

This bias in citation can be expected to show a similar trend for papers published in the middle - i.e. for those papers that were published in the middle 10 % of the publication time-line.

[1] (c) (i) The average in-degree of the papers with submission dates for the

- First 10 % of data is $\boxed{25.17}$, and
- Last 10 % of data is $\boxed{7.52}$

Steps taken to convert input data : First, the *cit-HepPh-dates.txt* file was read, and the data in it was stored in a pandas dataframe. The columns (containing the papers and the date on which they were published) were named in the dataframe, and duplicate entries for papers were removed. Then two new dataframes were created for storing the first 10 % and the last 10 % of the published papers from the *cit-HepPh-dates.txt* file.

Next the *cit-HepPh.txt* file was read, which contains the citation data for the papers published whose dates are mentioned in *cit-HepPh-dates.txt*, and this new data was stored in another pandas dataframe. A pandas series (similar to a dictionary in python) was created, and the citation counts for each paper was stored in it.

Finally the dataframes (in which the first 10 % and the last 10 % of the published papers data were stored) were recalled, and pandas series was used to fetch the number of citations for these papers.

[1] (c) (ii) ***Discussion of agreement of empirical values :*** It was observed that these empirical values did not agree to a great extent with the model estimated from the results of Problem 1 (b). While Problem 1 (b) generated citation counts of about 88 and about 0.18 for the first 10 % and last 10 % of published papers, the empirical values obtained from this dataset are about 25 and about 7 for the first 10 % and las t 10 % of published papers. This disagreement in values can be attributed to the fact that the preferential attachment mechanism does not capture what really transpires in real citation networks - all that Price's model captures are (a) a paper that is older will gather more citations than a paper that was recently published, and (b) a paper that has more citations will garner even more (compared to a paper that has lesser citations).

For instance, it does not capture the quality of paper - it merely assumes that 'higher' the number of citations, the 'better' or more preferred the paper for being cited for other papers. There can definitely be some papers that are relatively new (recently published) which could be worthy of as many citations as a paper that was published several years ago. Again, on similar lines, different 'types' of papers receive different number of citations - for example, review papers receive far greater number of citations than papers that have results. The PAM model has no procedure of capturing that.

Similarly, there are several other dynamics that occurs in real citation networks that the PAM model fails to account for, which leads to this discrepancy.

[1] **(d)** ***Ways in which the attachment mechanism is unrealistic for citation networks :***

1. **Knowledge of the all papers in the field :** When an author publishes a paper in a certain field for the first time, the author is unaware of all the papers in that field that have already been published. Price's model does not take this into account - it assumes that a vertex, when it enters a network, it knows about the existence of all the vertices in the network, and all the edges that exist between them. *To fix this, an author should select an arbitrary paper and cite a uniformly random paper listed in its bibliography. This is equivalent to choosing a random neighbor of a random vertex, which, in a random graph, leads to selecting a vertex with a probability proportional to its degree.*

2. **Quality of a paper :** Price's model does not consider in any way the quality of a paper. It directly assumes that the higher the number of citations a paper has, the better or more 'preferred' is the paper for garnering more citations. While this is not false by any means, there could be newer publications in a field that are worthy of gathering as many citations that an older paper garnered till the current point in time. In other words, the 'rich get richer' effect may be true, but some 'poor' papers maybe deserving enough (based on their quality) to get 'richer' than the 'richest' papers at the given point in time. *To fix this, a paper should have an attribute which captures its quality when it is published, and this attribute should be utilized to cite papers that are published later in the same field.*

3. **Time :** There could be a paper published in a field that had a large number of citations, but has not received a citation in the near past, i.e. the last time it was cited was a year ago, whereas all the other papers were cited less than a month ago. This means that the paper under consideration has become 'dormant', even though it may have had a large number of citations. *To fix this, an attribute should be assigned to each paper(vertex) which tracks the last time it received a citation, and information from this variable should be used to cite this paper - the more recently it was cited, the higher the probability that it gains more citations.* The converse is also true. There could be a paper that was published a while ago in a field which has not garnered a large number of citations, for no particular reason. Suddenly this paper might start gathering citations if a famous author cites it, or a group of scientists realize it's potential in a body of work that was conducted recently. Price's model does not account for such papers. *To fix this, a variable can be attributed to papers to track the density of citations received, or the number of citations received in a standard window of time. If that number is large, then it should be given more weight when being considered for being cited.*

4. **Different type of papers :** It is possible that papers with contrasting number of citations are equally 'important' for being considered for more citations, and Price's model does not capture that. For example, a review paper that has garnered 400 citations maybe as important as a results paper (results papers usually tend to garner lesser citations than review papers in similar amounts of time in a given field) that has gathered 20 citations.*To fix this, papers should have attributes that capture their type, and based on their type, the number of citations should be scaled to number that considers papers fairly for being cited from different groups.*

[1] (e) (i) ***Plot for ccdf using a variation of Price's Model of a citation network :*** Here, each time a new vertex joins the network, each of its c edges attaches to an existing vertex with equal probability. For choices of $c = 3$ and $n = 10^6$, shown below is a single figure showing the two complementary cumulative distribution functions when preferential attachment is considered, $\Pr(K \geq k_{in})$ (the ccdf) for network in-degree k_{in}, one for each choice of r = 1,4 and when there is no preferential attachment.

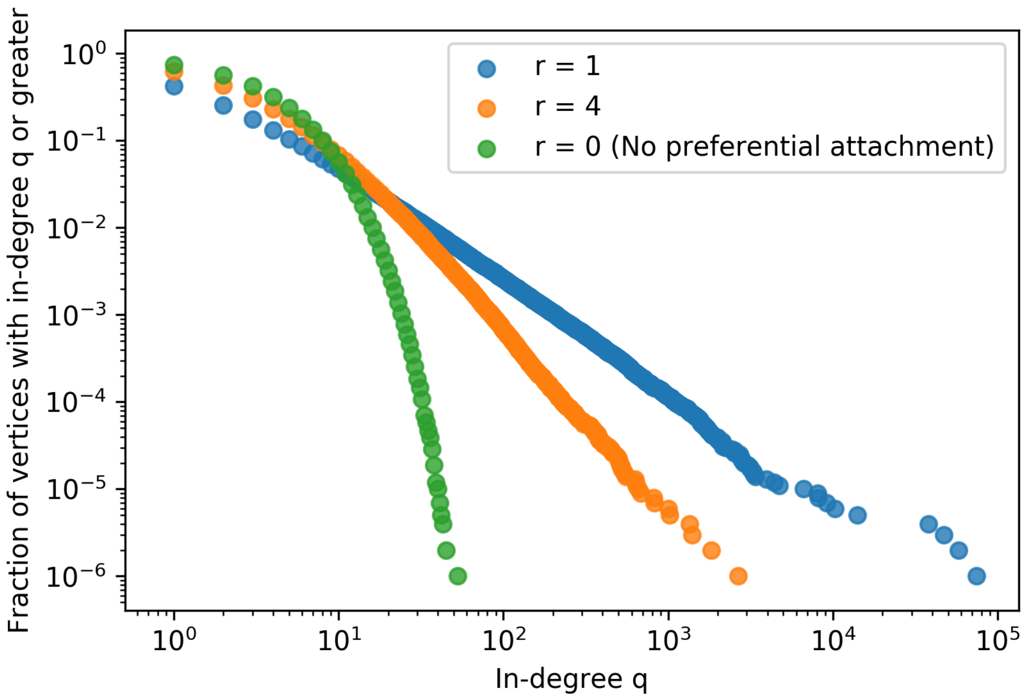

Figure 2: Plot of the complementary cumulative distribution functions versus the network in-degree for r = 1,4 when preferential attachment is considered **and** when there there is no preferential attachment.

[1] (e) (ii) ***Discussion :*** When we remove the preferential attachment part, there are two differences.

1. Firstly, we do not observe a linear curve any more for any value of r. We now have a fraction of vertices with in-degree q or greater that falls much more rapidly with in-degree q than a linear curve (on a log-log plot), and hence the relation between the fraction of vertices with in-degree q or greater and the in-degree q is not a power-law any more. It is a function that falls off faster than a power law does.

2. The other observation is that the vertices with the highest in-degree have a far lesser value of the in-degree than those for Problem 1 (a) (i.e. when we had the preferential attachment mechanism). This implies that 'rich get richer' effect is present to a far lesser degree here than when we had preferential attachment.

The code for [1] (a) (i) has been shown below.

```
import random
import matplotlib.pyplot as plt
import time

start_time = time.time()

# Initialize all the parameters for the network
n = 10**6 # Total number of nodes
#attach_cont = [i for i in range(1,5)] # Uniform attachment contribution
attach_cont = [1, 2, 3, 4]
c = 3 # k_out

ccdfs = []
indegrees = []
```

```
for r in attach_cont:

    p = c/(c+r) # Attachment probability
    #print(r,c,p)
    x = [0] * (c*n) # Store the complete network
    x[0:11] = [2, 3, 4, 1, 3, 4, 1, 2, 4, 1, 2, 3] # Initial clique seed
    x.pop()

    # Iterate through all the vertices
    for t in range(5,n+1):
        # Iterate through each out-edge
        for j in range(0,c):
            # Generate a random number between 0 and 1
            # and check if it is less than p
            if (random.uniform(0,1) < p):
                # choose an element uniformly at random
                # from the list of targets
                d = x[random.randint(0, c*(t-1)-1)]
                #print(d)
            else:
                # choose a vertex uniformly at random
                # from the set of all vertices
                d = random.randint(1, t-1)
                #print(d)
            x[c*(t-1) + j] = d

    # Initialize dictionary for counting the number of times
    # a node appears in the target list
    target_count = {}
    for i in x:
        if (i not in target_count):
            target_count[i] = 1
        else:
            target_count[i] += 1

    # Calculate the in-degree of nodes and store them in a dictionary
    ct = list(target_count.values())
    ct_dict = {}
    for c1 in ct:
        if (c1 not in ct_dict):
            ct_dict[c1] = 1
        else:
            ct_dict[c1] += 1

    # Calculate the cumulative complementary distribution function
    ccdf_dict = {}
    for key, value in ct_dict.items():
        #larger = [i for i in list(ct_dict.keys()) if key <= i]
        larger = []
        lt = list(ct_dict.keys())
        for i in lt:
            if (key <= i):
                larger.append(i)
        su = 0
        for j in larger:
            su += ct_dict[j]
        ccdf_dict[key] = su

    # Store the x and y values for plotting the ccdf
    x = []
    y = []
    for key, value in ccdf_dict.items():
        x.append(key)
        y.append(value/(n))

    # Store the graphs for different values of r
    indegrees.append(x)
    ccdfs.append(y)

# Plot the ccdf
plt.figure()
ax = plt.gca()
ax.set_xscale('log')
ax.set_yscale('log')
for x,y in zip(indegrees, ccdfs):
    ax.scatter(x,y, alpha = 0.8)
plt.legend(('r = 1', 'r = 2', 'r = 3', 'r = 4'))
ax.set_xlabel('In-degree q')
ax.set_ylabel('Fraction of vertices with in-degree q or greater')
plt.savefig('ccdf_vs_indegree_all-r-2', dpi = 300)
plt.show()

elapsed_time = time.time() - start_time
print(elapsed_time, 'seconds' )
```

The code for [1] (b) (i) has been shown below.

```
import random
import time
import numpy as np

start_time = time.time()

# Initialize all the parameters for the network
n = 10**6 # Total number of nodes
simulations = [i for i in range(1,11)]
c = 12 # k_out
r = 5

p = c/(c+r) # Attachment probability
x = [0] * (c*n) # Store the complete network
x[0:11] = [2, 3, 4, 1, 3, 4, 1, 2, 4, 1, 2, 3] # Initial clique seed
x.pop()

all_first_ten = []
all_last_ten = []

for sim in simulations:


    x = [0] * (c*n) # Store the complete network
    x[0:11] = [2, 3, 4, 1, 3, 4, 1, 2, 4, 1, 2, 3] # Initial clique seed
    x.pop()
    # Iterate through all the vertices
    for t in range(5,n):
        # Iterate through each out-edge
        for j in range(0,c):
            # Generate a random number between 0 and 1
            # and check if it is less than p
            if (np.random.rand() < p):
                # choose an element uniformly at random
                # from the list of targets
                d = x[random.randint(0, len(x)-1)]
                #print(d)
            else:
                # choose a vertex uniformly at random
                # from the set of all vertices
                d = random.randint(1, t-1)
                #print(d)

            #x[c*(t-1) + j] = d
            x[c*(t-1) + j] = d

    # Initialize dictionary for counting the number of times
    # a node appears in the target list
    target_count = {}
    for i in x:
        if (i not in target_count):
            target_count[i] = 1
        else:
            target_count[i] += 1

    # Compute the average number of citations for the
    # first ten percent of the publications
    sum_first_ten = 0
    for i in range(100000):
        if i in target_count:
            sum_first_ten += target_count[i]

    avg_first_ten = sum_first_ten/100000

    # Compute the average number of citations for the
    # last ten percent of the publications
    sum_last_ten = 0
    for i in range(900000,1000000):
        if i in target_count:
            sum_last_ten += target_count[i]

    avg_last_ten = sum_last_ten/100000

    all_first_ten.append(avg_first_ten)
    all_last_ten.append(avg_last_ten)


print(sum(all_first_ten)/len(all_first_ten))
print(sum(all_last_ten)/len(all_last_ten))
print()

elapsed_time = time.time() - start_time

print(elapsed_time, 'seconds')
```

The code for [1] (c) (i) has been shown below.

```
import pandas as pd

# Read the data and store it in a pandas dataframe
data = pd.read_csv('cit-HepPh-dates.txt', skiprows = [0],
                   sep='\t', header=None)
# Name the columns
data.columns = ['Paper', 'Date']
#n1 = data.shape[0]
#print('Before', n1)
```

```
# Remove duplicate entries
data.drop_duplicates(subset = 'Paper', keep = False, inplace = True)
n = data.shape[0]
#print('After', n)

first_ten = int(n/10)

# Store the first 10 % of published papers in a new dataframe,
# name the columns and make a new list for counting the citations
data_first = data[:first_ten]
data_first.columns = ['Col-1', 'Col-2']
average_first = []

# Store the last 10 % of published papers in a new dataframe,
# name the columns and make a new list for counting the citations
data_last = data[-first_ten:]
data_last.columns = ['Col-1', 'Col-2']
average_last = []

# Read and store the edge-list dataset into a new pandas dataframe
edge_data = pd.read_csv('cit-HepPh.txt', skiprows = [0,1,2,3],
                        sep = '\t', header = None)
edge_data.columns = ['Tail', 'Head']
n_edges = edge_data.shape[0]
cit_count = {}

for index, rows in edge_data.iterrows():
    if (rows['Head'] not in cit_count):
        cit_count[rows['Head']] = 1
    else:
        cit_count[rows['Head']] += 1

# Make a pandas series for citation counts from edge-list dataset
citations = edge_data['Head'].value_counts()

# Fetch and store the number of citations for the first 10%
count = 0
for index, rows in data_first.iterrows():
    ids = rows['Col-1']
    if (ids in citations.keys()):
        count = citations[ids]
        average_first.append(count)

# Fetch and store the number of citations for the last 10%
count = 0
for index, rows in data_last.iterrows():
    ids = rows['Col-1']
    if (ids in citations.keys()):
        count = citations[ids]
        average_last.append(count)

print('Average in first ten :', sum(average_first)/len(average_first))
print('Average in last ten :', sum(average_last)/len(average_last))
```

The code for [1] (e) (i) has been shown below.

```
import random
import matplotlib.pyplot as plt
import time
```

```
start_time = time.time()

# Initialize all the parameters for the network
n = 10**6 # Total number of nodes
attach_cont = [1,4] # Uniform attachment contribution
c = 3 # k_out

ccdfs = []
indegrees = []

#p = c/(c+r) # Attachment probability
x = [0] * (c*n) # Store the complete network
x[0:11] = [2, 3, 4, 1, 3, 4, 1, 2, 4, 1, 2, 3] # Initial clique seed
x.pop()

#-----------------------------------------------------------------

for r in attach_cont:
    p = c/(c+r) # Attachment probability
    x = [0] * (c*n) # Store the complete network
    x[0:11] = [2, 3, 4, 1, 3, 4, 1, 2, 4, 1, 2, 3] # Initial clique seed
    x.pop()

    # Iterate through all the vertices
    for t in range(5,n+1):
        # Iterate through each out-edge
        for j in range(0,c):
            # Generate a random number between 0 and 1
            # and check if it is less than p
            if (random.uniform(0,1) < p):
                # choose an element uniformly at random
                # from the list of targets
                d = x[random.randint(0, c*(t-1)-1)]
                #print(d)
            else:
                # choose a vertex uniformly at random
                # from the set of all vertices
                d = random.randint(1, t-1)
                #print(d)
            x[c*(t-1) + j] = d

    # Initialize dictionary for counting the number of times
    # a node appears in the target list
    target_count = {}
    for i in x:
        if (i not in target_count):
            target_count[i] = 1
        else:
            target_count[i] += 1

    # Calculate the in-degree of nodes and store them in a dictionary
    ct = list(target_count.values())
    ct_dict = {}
    for c1 in ct:
        if (c1 not in ct_dict):
            ct_dict[c1] = 1
        else:
            ct_dict[c1] += 1

    # Calculate the cumulative complementary distribution function
    ccdf_dict = {}
    for key, value in ct_dict.items():
        #larger = [i for i in list(ct_dict.keys()) if key <= i]
        larger = []
        lt = list(ct_dict.keys())
        for i in lt:
            if (key <= i):
                larger.append(i)
        su = 0
        for x in larger:
            su += ct_dict[x]
        ccdf_dict[key] = su

    # Store the x and y values for plotting the ccdf
    x = []
    y = []
    for key, value in ccdf_dict.items():
        x.append(key)
        y.append(value/(n))

    # Store the graphs for different values of r
    indegrees.append(x)
    ccdfs.append(y)

#-----------------------------------------------------------------------

# No preferential attachment part
x = [0] * (c*n) # Store the complete network
x[0:11] = [2, 3, 4, 1, 3, 4, 1, 2, 4, 1, 2, 3] # Initial clique seed
x.pop()

# Iterate through all the vertices
for t in range(5,n+1):
    # Iterate through each out-edge
    for j in range(0,c):
        # choose a vertex uniformly at random
        # from the set of all vertices
        d = random.randint(1, t-1)
        x[c*(t-1) + j] = d

# Initialize dictionary for counting the number of times
# a node appears in the target list
target_count = {}
for i in x:
    if (i not in target_count):
        target_count[i] = 1
    else:
        target_count[i] += 1

# Calculate the in-degree of nodes and store them in a dictionary
ct = list(target_count.values())
ct_dict = {}
for c1 in ct:
    if (c1 not in ct_dict):
        ct_dict[c1] = 1
    else:
        ct_dict[c1] += 1

# Calculate the cumulative complementary distribution function
ccdf_dict = {}
for key, value in ct_dict.items():
    #larger = [i for i in list(ct_dict.keys()) if key <= i]
    larger = []
    lt = list(ct_dict.keys())
    for i in lt:
        if (key <= i):
            larger.append(i)
    su = 0
    for x in larger:
        su += ct_dict[x]
    ccdf_dict[key] = su

# Store the x and y values for plotting the ccdf
x = []
y = []
for key, value in ccdf_dict.items():
    x.append(key)
    y.append(value/(n))

# Store the graphs for different values of r
indegrees.append(x)
ccdfs.append(y)

#-------------------------------------------------------------------------

# Plot the ccdf
plt.figure()
ax = plt.gca()
ax.set_xscale('log')
ax.set_yscale('log')
for x,y in zip(indegrees, ccdfs):
    ax.scatter(x,y, alpha = 0.8)
plt.legend(('r = 1', 'r = 4', 'r = 0 (No preferential attachment)'))
ax.set_xlabel('In-degree q')
ax.set_ylabel('Fraction of vertices with in-degree q or greater')
plt.savefig('ccdf_vs_indegree_all-r_no-pref-attach-2', dpi = 300)
plt.show()

elapsed_time = time.time() - start_time

print(elapsed_time, 'seconds' )
```