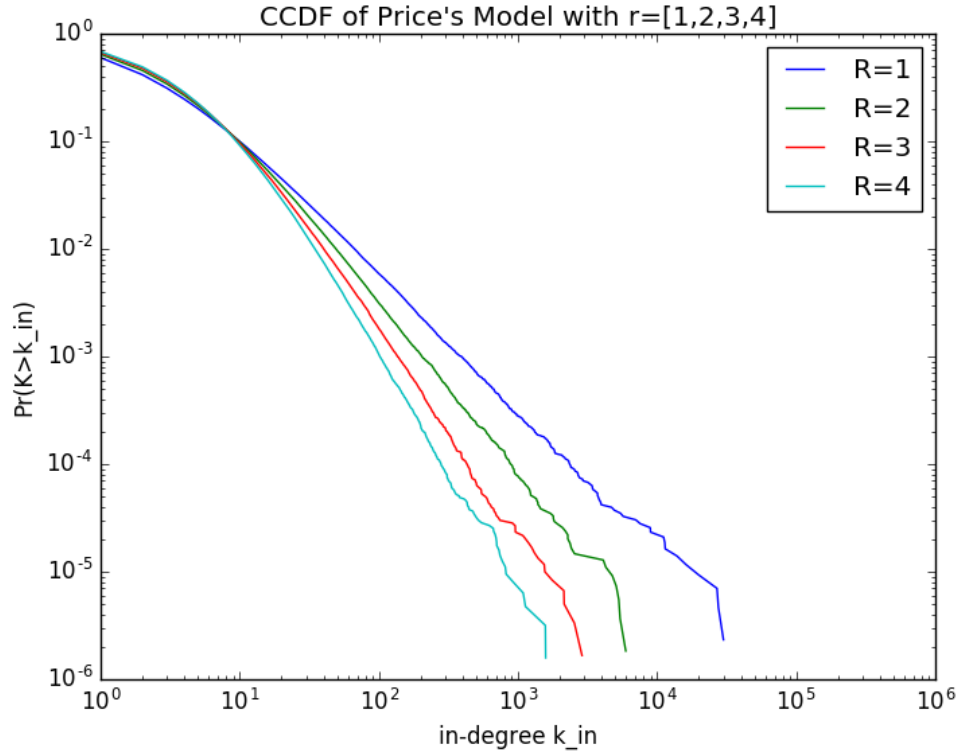


1. a.



At low  $k_{in}$  values less than 10, distributions of Price's model with varying  $R$  values are very similar. In this area of the distribution, in-degree is primarily driven by random attachment. At in-degree greater than 10, the  $R$  value becomes more important in determining in-degree, which can be seen in the separate distributions. As  $R$  increases, the affect of preferential attachment decreases, causing a narrower distribution in in-degree. After the 'shoulder' driven by random attachment, lower  $R$  values cause a more gradual drop-off in in-degree, reflecting a wider distribution. Without preferential attachment, degree distribution would be much narrower, with a steeper drop-off in the CCDF plot (as seen in question 1.e.)

The opposite effect can be seen, to a small degree, for  $\Pr(K > k_{in})$  where  $k_{in}$  is less than 10. At lower  $R$  values, the fraction of nodes with degree greater than 1 to 10 is higher than larger  $R$  values. This is caused by the larger effect of random attachment. Nodes that are added to the network towards the end of the growth period have an increased chance of

being linked to by even later additions, as opposed to the preferential attachment model, which would give these 'late-comer' nodes a small chance of having an increased in-degree. This explains the small but consistent trend at in-degree = 0. As the R value decreases, the fraction of nodes with in-degree = 0 also decreases.

1.b. Results were obtained using the simulation designed for 1.a. above, with  $c=12$ ,  $R=5$ , and  $n=10^6$ . (i) Averaging over 100 iterations, the average in-degree of the first 10% (100,000) of papers published was determined to be 81.34. The range of in-degree for the first 10% of papers was quite large, with the largest average in-degree of 38,083.27, and the smallest in-degree of 17.11.

(ii). Average in-degree for the last 10% of nodes added to the network was 0.187. The largest average in-degree was 0.63, with many nodes having in-degree of zero.

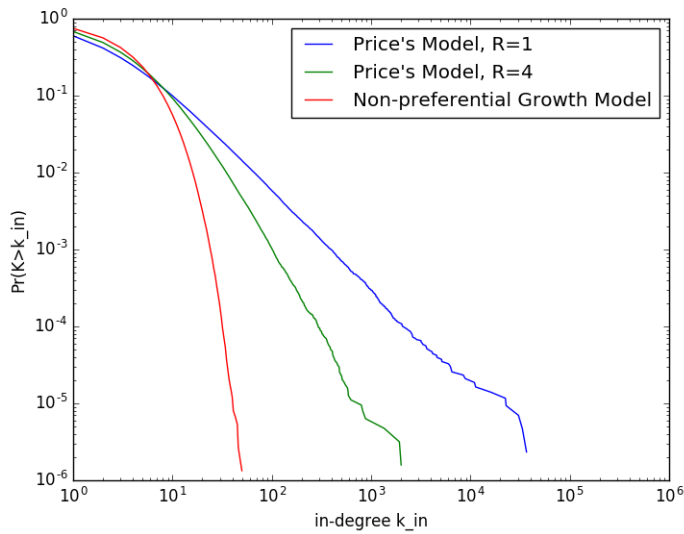
To examine the first-mover advantage, we can examine the results for the first 10% of papers more carefully. The highest average in-degree is over 38,000, but moving down the sorted list of average in-degree just 13 places gives an average in-degree of 20,080. By position 587, average in-degree is less than 1000. The first-mover advantage definitely exists, and is limited to the small fraction of nodes added at the beginning of graph construction. A similar examination of results for the last 10% of papers published reveals a very small range, as previously mentioned. A paper published in the last 90th percentile has very little advantage over even the very last paper published.

1.c. Average in-degree for the first 10% of papers published in the "arXiv citation networks (1993-2003)" was determined to be 23.62, while the average in-degree for the last 10% of papers published was 14.36. These results do not agree with results from 1.b. While there are more papers with relatively high in-degree in the first 10% (over 100, for example), there are also many papers with low (less than 10) in-degree. The extreme first-mover advantage seen in 1.b. is not present in this real network. Results for the last 10% of papers published also disagree with 1.b. There are very few papers with 0 in-degree in the last 10% of this network, in contrast to the many found in the simulation conducted in 1.b.

A possible explanation for absence of the extreme first-mover advantage seen in 1.b. could be that authors were not forced to rely solely on information and ideas presented in the early papers. For example, the 10th paper published could be drawing on outside sources and independently formulating conclusions from the authors of the earlier papers. In a new field, particularly, this may be common. Many authors, while working in the same new field, draw from various sources. This would reduce the first-mover advantage as other authors continue to publish. Subsequent authors can also choose among the previously-published papers of highest quality. This would give a small range of papers a high in-degree, rather than just a few early-movers.

1.d. Ways that the preferential attachment mechanism is unrealistic are the extreme advantage given to first-movers, the extreme penalty given to late-comers, and the disregard for quality of work in a given paper, regardless of time published. To analyze the first weakness of the preferential attachment model, the extreme first-mover advantage, we can perform analysis similar to 1.c. above. A preferential attachment simulation shows a very strong first-mover advantage, while analysis of a real network does not bear this out. There is definitely higher in-degree in earlier-published papers in the real network, but nowhere near as large as that predicted by simulation. The extreme penalty for late-comers can be examined in a similar manner. Simulations predict average in-degree of late-comers to be much less than 1, while real network examination in 1.c. does not agree. To determine if quality of work, regardless of paper publishing date, one could look for outliers in both large and small expected in-degree. The existence of a paper published in the very early stages of network growth, but with very low in-degree, could be an indicator of poor methodology or experiment design, among other things. Similarly, a paper with very high in-degree among the last papers published could be an indicator of a breakthrough, interesting conclusions, etc. that allows the paper to gather many more citations than the preferential model would predict.

1.e. Removing the preferential attachment mechanism from our simulation used for 1.a. above acts to drastically reduce the distribution of node in-degree. As seen in 1.a, the distribution of in-degrees narrows as  $R$  increases. When preferential attachment is removed completely, we see the same 'shoulder' present in Price's model, but after that, the fraction of nodes with larger in-degree drops off very quickly to zero. This plot completely lacks the power-log curve shown by Price's model. The fraction of nodes with in-degree zero is smaller, since even the very last papers published have a higher chance of being cited without the preferential attachment mechanism.



Code for 1.a.

```

1  # Donovan Guelde
2  # CSCI-5352
3  # PS6 Q.1.a
4
5  import numpy as np
6  import matplotlib.pyplot as plt
7
8
9
10 def growNetwork(r,n,pr,c):
11
12     vertexLabellist=np.full(n*c,-1)
13     adjacencyList=np.zeros(shape=(n,c))
14     adjacencyList.fill(-1)
15     numberVertexes=0
16     numberEdges=0
17     iterations=1
18     #make a 'seed' graph, c+1 nodes to simulate growth from 0 nodes
19     #node 0 has no outgoing edges, node 1 points to 0, node 2 points to 0 and 1,
20     #node 3 points to nodes 0,1,2, etc. Node[c+1] now has c outgoing edges
21     for index in range(0,c):
22         for index2 in range(index+1,c+1):
23             adjacencyList[index2][index]=index
24             vertexLabellist[numberEdges]=index
25             numberEdges+=1
26     numberVertexes=c+1 #we preseeded |c+1| vertexes
27     for index in range(numberVertexes,n): #add remaining vertexes to network
28         chosen=[]

```

```

29         tempList=vertexLabelList[0:numberEdges]
30         coinFlips=np.random.random(c)
31         for index2 in range(0,c): #each new vertex has c out-degree
32             if coinFlips[index2]<pr: #choose in proportion to in-degree
33                 edgePointsTo=np.random.choice(tempList)
34                 while (edgePointsTo in chosen or edgePointsTo==-1):
35                     edgePointsTo=np.random.choice(tempList)
36             else: #randomly select
37                 edgePointsTo=np.random.randint(0,high=numberVertexes-1)
38                 while(edgePointsTo in chosen):
39                     edgePointsTo=np.random.randint(0,high=numberVertexes-1)
40             chosen.append(edgePointsTo)
41         counter=0
42         numberVertexes+=1
43         for item in chosen: #add new edges to list
44             adjacencyList[index][counter]=item
45             vertexLabelList[numberEdges]=item
46             counter+=1
47             numberEdges+=1
48     return adjacencyList[1:]
49
50
51
52
53 def main():
54     #parameters per assignment
55     c=3
56     N=1000000
57     R=[1.,2.,3.,4.]
58     XaxisResults=[]
59     YaxisResults=[]
60     #bigger r --> lower pr --> higher chance of random assignment of new edges --> more uniform distribution
61     for r in R:
62         pr=c/(c+r) #probability of attaching to node in proportion of in-degree
63         n=N
64         fileName = '1a_plot.png'.format(str(r))
65
66         adjacencyList=growNetwork(r,n,pr,c)
67         nodes,indegree=np.unique(adjacencyList,return_counts=True) #
68         uniqueValues=np.unique(indegree)
69         indegreeList=np.zeros(n)
70         counter=0
71         for item in nodes:
72             if(item!=-1):
73                 indegreeList[item]=indegree[counter]
74                 counter+=1
75
76         xAxis=np.unique(indegreeList) #observed degree on x axis
77         XaxisResults.append(xAxis)
78         yAxis=np.zeros(len(xAxis))
79         counter=0
80         for item in uniqueValues:
81             yAxis[counter]=np.count_nonzero(indegreeList>=item) #fraction (nodes indegree >= indegree k)
82             counter+=1
83         yAxis=yAxis/float(np.amax(yAxis)) #regularize as a fraction < 1
84         YaxisResults.append(yAxis)
85
86     labels=['R=1', 'R=2', 'R=3', 'R=4']

```

```

87     plt.title('CCDF of Price\'s Model with r=[1,2,3,4]')
88     plt.xlabel('in-degree k_in')
89     plt.ylabel('Pr(K>k_in)')
90     plt.xlim(1,n)
91     for index in range(0,len(R)):
92         plt.loglog(XaxisResults[index],YaxisResults[index],label=labels[index])
93     plt.legend(loc='upper right')
94     plt.savefig(fileName)
95     plt.close()
96
97     main()

```

Code for 1.b.

```

1  # Donovan Guelde
2  # CSCI-5352
3  # PS6 Q.1.b
4
5  import numpy as np
6  import matplotlib.pyplot as plt
7  import time
8
9
10
11 def growNetwork(r,n,pr,c):
12
13     vertexLabelList=np.full(n*c,-1)
14     adjacencyList=np.zeros(shape=(n,c))
15     adjacencyList.fill(-1) #-1 signifies no edge
16     numberVertexes=0
17     numberEdges=0
18
19     #make a 'seed' graph, c+1 nodes to simulate growth from 0 nodes
20     #node 0 has no outgoing edges, node 1 points to 0, node 2 points to 0 and 1,
21     #node 3 points to nodes 0,1,2, etc. Node[c+1] now has c outgoing edges
22     for index in range(0,c):
23         for index2 in range(index+1,c+1):
24             adjacencyList[index2][index]=index
25             vertexLabelList[numberEdges]=index
26             numberEdges+=1
27     numberVertexes=c+1 #we preseeded |c+1| vertices
28     for index in range(numberVertexes,n): #add remaining vertexes to network
29         chosen=[]
30         tempList=vertexLabelList[0:numberEdges]
31         coinFlips=np.random.random(c)
32         for index2 in range(0,c): #each new vertex has c out-degree
33             if coinFlips[index2]<pr: #choose in proportion to in-degree
34                 edgePointsTo=np.random.choice(tempList)
35                 while (edgePointsTo in chosen or edgePointsTo==-1):
36                     edgePointsTo=np.random.choice(tempList)
37             else: #randomly select
38                 edgePointsTo=np.random.randint(0,high=numberVertexes-1)
39                 while (edgePointsTo in chosen):
40                     edgePointsTo=np.random.randint(0,high=numberVertexes-1)
41             chosen.append(edgePointsTo)
42         counter=0
43         numberVertexes+=1
44         for item in chosen: #add new edges to list

```

```

45         adjacencyList[index][counter]=item
46         vertexLabelList[numberEdges]=item
47         counter+=1
48         numberEdges+=1
49
50     return adjacencyList
51
52
53
54
55 def main():
56     #parameters per assignment
57     c=12
58     n=1000000
59     r=5.
60     iterations=100 ##
61     fileName = '1b-{}.txt'.format(str(iterations))
62     pr=c/(c+r) #probability of attaching to node in proportion of in-degree
63     results=np.zeros((iterations,n*.2))
64
65
66
67
68     for iteration in xrange(iterations):
69
70         start=time.time()
71         adjacencyList=growNetwork(r,n,pr,c).astype(int)
72         adjacencyList=adjacencyList[adjacencyList>=0]
73         adjacencyList=np.sort(adjacencyList,axis=None)
74
75         for index in xrange(len(adjacencyList)):
76             v = adjacencyList[index]
77
78             if ( v < int(n*.1)):
79                 results[iteration][v]+=1
80
81             if (v >= int(n*.9)):
82                 results[iteration][v-n]+=1
83
84
85         print time.time()-start
86
87     average=np.average(results,axis=0)
88     np.savetxt(fileName,average,fmt='%1.4f')
89
90
91
92
93
94 main()

```

Code for 1.c.

```

1  # Donovan Guelde
2  # CSCI 5352 PS6 q1c
3
4  import numpy as np
5  from igraph import *

```

```

6  from sets import Set
7
8  def readFile():
9      g = Graph(directed=True)
10
11      with (open('cit-HepPh-dates.txt','r')) as f:
12          next(f) #skip header row
13          nodesByDate=[]
14          for line in f:
15              line=line.split()
16              node,date=line[0],line[1]
17              nodesByDate.append(node)
18
19
20
21
22      edgeList=[]
23      with (open('cit-HepPh.txt','r')) as f:
24
25          nodes=Set()
26          f.seek(0,0)
27          next(f)
28          next(f)
29          next(f)
30          next(f)
31          for line in f:
32              line=line.split()
33              node,neighbor=line[0],line[1]
34              if node not in nodes:
35                  nodes.add(node)
36              if neighbor not in nodes:
37                  nodes.add(neighbor)
38
39              edgeList.append((node,neighbor))
40
41      for item in nodes:
42          g.add_vertex(item)
43      g.add_edges(edgeList)
44      #how many vertices have date info?
45      count=0.
46      for item in nodes:
47          if item in nodesByDate:
48              count+=1
49
50      return g,nodesByDate,nodes,count
51
52  def main():
53      g,nodesByDate,nodes,haveDateInfo=readFile()
54      interval=int(.1*haveDateInfo)
55      print interval
56      print nodesByDate[0]
57      firstTenPercent=[]
58      count=0
59      counter=0
60      firstTenTotal=0.
61      while(count<interval):
62          try:
63              firstTenTotal+= Graph.degree(g,nodesByDate[counter],mode='in')

```



```

64         node= nodesByDate[counter]
65         degree= Graph.degree(g,nodesByDate[counter],mode='in')
66         firstTenPercent.append((node,degree))
67         count+=1
68     except ValueError:
69         pass
70     counter+=1
71     print "first 10% average:",firstTenTotal/interval
72
73     counter=int(haveDateInfo)
74     sum=0.
75     count=0
76     lastTenPercent=[]
77     lastTenTotal=0.
78     while(count<interval):
79         try:
80             lastTenTotal+= Graph.degree(g,nodesByDate[counter],mode='in')
81             node= nodesByDate[counter]
82             degree= Graph.degree(g,nodesByDate[counter],mode='in')
83             lastTenPercent.append((node,degree))
84             count+=1
85         except ValueError:
86             pass
87         counter-=1
88     print "last 10% average:",lastTenTotal/interval
89
90     with(open('1cResults.txt','w')) as f:
91         f.write('first ten percent:\n')
92         f.write('\n'.join('%s %s' % x for x in firstTenPercent))
93
94         f.write("total: "+str(firstTenTotal)+'\n')
95         f.write("Average: "+str(firstTenTotal/interval))
96         f.write('\n\n')
97         f.write('last ten percent:\n')
98         f.write('\n'.join('%s %s' % x for x in lastTenPercent))
99         f.write('total: '+str(lastTenTotal)+'\n')
100        f.write("Average: "+str(lastTenTotal/interval))
101
102
103
104
105
106
107    main()

```

Code for 1.e.

```

1  # Donovan Guelde
2  # CSCI-5352
3  # PS6 Q.1.e
4
5  import numpy as np
6  import matplotlib.pyplot as plt
7
8
9
10 def growNetwork(r,n,pr,c):
11

```

```

12     vertexLabelList=np.full(n*c,-1)
13     adjacencyList=np.zeros(shape=(n,c))
14     adjacencyList.fill(-1)
15     numberVertexes=0
16     numberEdges=0
17     iterations=1
18     #make a 'seed' graph, c+1 nodes to simulate growth from 0 nodes
19     #node 0 has no outgoing edges, node 1 points to 0, node 2 points to 0 and 1,
20     #node 3 points to nodes 0,1,2, etc. Node[c+1] now has c outgoing edges
21     for index in range(0,c):
22         for index2 in range(index+1,c+1):
23             adjacencyList[index2][index]=index
24             vertexLabelList[numberEdges]=index
25             numberEdges+=1
26     numberVertexes=c+1 #we preseeded |c+1| vertexes
27     for index in range(numberVertexes,n): #add remaining vertexes to network
28         chosen=[]
29         coinFlips=np.random.random(c)
30         tempList=vertexLabelList[0:numberEdges]
31         for index2 in range(0,c): #each new vertex has c out-degree
32             if coinFlips[index2]<pr: #choose in proportion to in-degree
33                 edgePointsTo=np.random.choice(tempList)
34                 while (edgePointsTo in chosen or edgePointsTo==-1):
35                     edgePointsTo=np.random.choice(tempList)
36             else: #randomly select
37                 edgePointsTo=np.random.randint(0,high=numberVertexes-1)
38                 while(edgePointsTo in chosen):
39                     edgePointsTo=np.random.randint(0,high=numberVertexes-1)
40             chosen.append(edgePointsTo)
41         counter=0
42         numberVertexes+=1
43         for item in chosen: #add new edges to list
44             adjacencyList[index][counter]=item
45             vertexLabelList[numberEdges]=item
46             counter+=1
47         numberEdges+=1
48     return adjacencyList[1:]
49
50
51 def growNetworkNonpreferential(n,c):
52     vertexLabelList=np.full(n*c,-1)
53     adjacencyList=np.zeros(shape=(n,c))
54     adjacencyList.fill(-1)
55     numberVertexes=0
56     numberEdges=0
57     iterations=1
58     #make a 'seed' graph, c+1 nodes to simulate growth from 0 nodes
59     #node 0 has no outgoing edges, node 1 points to 0, node 2 points to 0 and 1,
60     #node 3 points to nodes 0,1,2, etc. Node[c+1] now has c outgoing edges
61     for index in range(0,c):
62         for index2 in range(index+1,c+1):
63             adjacencyList[index2][index]=index
64             vertexLabelList[numberEdges]=index
65             numberEdges+=1
66     numberVertexes=c+1 #we preseeded |c+1| vertexes
67     for index in range(numberVertexes,n): #add remaining vertexes to network
68         chosen=[]
69         tempList=vertexLabelList[0:numberEdges]

```

```

70         for index2 in range(0,c): #each new vertex has c out-degree
71             edgePointsTo=np.random.randint(0,high=numberVertexes-1)
72             while(edgePointsTo in chosen):
73                 edgePointsTo=np.random.randint(0,high=numberVertexes-1)
74             chosen.append(edgePointsTo)
75         counter=0
76         numberVertexes+=1
77         for item in chosen: #add new edges to list
78             adjacencyList[index][counter]=item
79             vertexLabelList[numberEdges]=item
80             counter+=1
81             numberEdges+=1
82     return adjacencyList[1:]
83
84
85 def main():
86     #parameters per assignment
87     c=3
88     N=1000000
89     XaxisResults=[]
90     YaxisResults=[]
91
92     R=[1.,4.]
93     #bigger r --> lower pr --> higher chance of random assignment of new edges --> more uniform distribution
94     for r in R:
95         pr=c/(c+r) #probability of attaching to node in proportion of in-degree
96         n=N
97         fileName = '1e_plot_r={}.png'.format(str(r))
98
99         adjacencyList=growNetwork(r,n,pr,c)
100        nodes,indegree=np.unique(adjacencyList,return_counts=True) #
101        uniqueValues=np.unique(indegree)
102        indegreeList=np.zeros(n)
103        counter=0
104        for item in nodes:
105            if(item!=-1):
106                indegreeList[item]=indegree[counter]
107                counter+=1
108
109        xAxis=np.unique(indegreeList) #observed degree on x axis
110        XaxisResults.append(xAxis)
111        yAxis=np.zeros(len(xAxis))
112        counter=0
113        for item in uniqueValues:
114            yAxis[counter]=np.count_nonzero(indegreeList>=item) #fraction (nodes indegree >= indegree k)
115            counter+=1
116        yAxis=yAxis/float(np.amax(yAxis)) #regularize as a fraction < 1
117        YaxisResults.append(yAxis)
118
119        adjacencyList=growNetworkNonpreferential(n,c) #non-preferential attachment graph
120        nodes,indegree=np.unique(adjacencyList,return_counts=True) #
121        uniqueValues=np.unique(indegree)
122        indegreeList=np.zeros(n)
123        counter=0
124        for item in nodes:
125            if(item!=-1):
126                indegreeList[item]=indegree[counter]
127                counter+=1

```

```

128
129     xAxis=np.unique(indegreeList) #observed degree on x axis
130     XaxisResults.append(xAxis)
131     yAxis=np.zeros(len(xAxis))
132     counter=0
133     for item in uniqueValues:
134         yAxis[counter]=np.count_nonzero(indegreeList>=item) #fraction (nodes indegree >= indegree k)
135         counter+=1
136     yAxis=yAxis/float(np.amax(yAxis)) #regularize as a fraction < 1
137     YaxisResults.append(yAxis)
138
139     labels=['Price\'s Model, R=1','Price\'s Model, R=4','Non-preferential Growth Model']
140     plt.xlabel('in-degree k_in')
141     plt.ylabel('Pr(K>k_in)')
142     plt.xlim(1,n)
143     for index in range(0,len(R)+1):
144         plt.loglog(XaxisResults[index],YaxisResults[index],label=labels[index])
145
146     plt.legend(loc='upper right')
147     plt.savefig(fileName)
148     plt.close()
149
150     main()

```