IIA SF5 Modelling Independent Cascade Model in Undirected Scale Free Network

htl28

June 2024

Abstract

The report investigates influence spread in an undirected scale-free network with the Independent Cascade Model as the spreading mechanism. We generate the networks and simulations using Python, then vary the network and simulation parameters to observe the average influence spread. There was positive correlation between average influence spread and initial seed size. With influence maximization in mind, we delve into different seeding protocols and their impact on average influence spread. We then discuss the impact with the cost of computing the initial seeds considered.The degree seeding protocol seemed to have the best reward-to-effort ratio, out of the random, degree, fixed degree, neighbour degree and fixed neighbour degree seeding protocols.

1 Introduction

Social network analysis is an extremely valuable research tool when applied in the field of social media marketing, as discovered by Lewis, 2023. The data reveals important users and communities through which information could be diffused, which is highly useful in navigating economic issues and planning social media marketing strategy. We study network dynamics to helps us understand how information, behaviours or influence spread through networks. A notable example of this is the phenomenon of information cascades, where individuals make the same decision in a sequential fashion, with probability influenced by observing others' choices.

'The Independent Cascade Model(ICM) is a type of information diffusion model where the information/influence propagates in a stochastic manner.' The ICM is chosen over the Linear Threshold Model (LTM) because it is simpler and easier to analyse, and it is better for situations where influence is fleeting or one-shot, like clicking the link on a product placement post from an influencer. Barabási, 1999 found that some social and biological networks can be described as 'scale-free', which are a class of networks that exhibit a power-law degree distribution. So in this technical report, we focus on the application of ICM within the context of an undirected (for simplicity) scale-free network generated using the Barabasi-Albert (BA) model, to closely simulate influence diffusion in real-life social networks.

As discovered by Lewis, 2023, the limitations of network analysis are the time-consuming factor and the overwhelming amount of information for brand communities trying to find key opinion leaders as advocated for the brand. At the end of the day, the typical goal is influence maximization. Therefore, we investigate the effectiveness of different seeding protocol (analogous to identifying the best key opinion leaders/social media influencers/brand ambassadors) and the cost required to identify such optimal set of seeds to determine if the selection process is worth it.

2 Methods

All the simulations were performed with Python 3.

2.1 Barabasi-Albert Network

'The Barabasi-Albert (BA) model is an algorithm for generating random scale-free networks using a preferentail attachment mechanism' according to Molak, 2017. The characteristic of a scale-free network is the presence of few nodes that are highly connected to other nodes in the network. This give the degree distribution a long tail, or we say that it has a power-law degree distribution that can be represented by

$$P_{deg}(k) \propto k^{-\gamma} \tag{1}$$

where γ is some exponent. At all scales, the power laws have the same functional form, so is therefore useful for modelling real networks of various population sizes.
One way to generate scale-free network is using a preferential attachment algorithm. In the BA model, the preferentail attachment is such that the probability that a new node connects to an existing node i is

$$\Pi(k_i) = \frac{k_i}{\sum_j k_j} \tag{2}$$

where k_i denotes the degree of node i.
The network parameters required when building a BA model is n, the final number of nodes in the network and m, the number of new edges every new nodes creates, and is also the initial number of nodes.
There are two ways to generate a BA Network Model in Python 3.

1. Code the classes and methods from scratch
2. Using **'networkx'** library

After comparing time taken to generate a network using both methods, it was discovered that using **'networkx'** is a one-liner which get the generation done many times faster than coding from scratch, so using **'networkx'** it is.
For the Barabasi-Albert Network, $\gamma = 3$ for almost all parameters, due to the specific growth and attachment rules. We therefore utilise this model throughout the investigation to keep γ constant.
According to BARABÁSI, n.d., the exact degree distribution of the BA model is

$$P_{deg}(k) = p_k = \frac{2m(m+1)}{k(k+1)(k+2)} \tag{3}$$

In the limit as N is large, the expected average degree is

$$\begin{aligned} lim_{N\to\infty} < k > &= \sum_{k=m}^{\infty} kp_k \\ &= \sum_{k=m}^{\infty} \frac{2m(m+1)}{k(k+1)(k+2)} \\ &= \sum_{k=m}^{\infty} (\frac{2m(m+1)}{k+1} - \frac{2m(m+1)}{k+2}) \quad = \frac{2m(m+1)}{m+1} \\ &= 2m \end{aligned} \tag{4}$$

Therefore when picking a desired mean degree,$< k >$, for the network, simply adjust the parameter m.

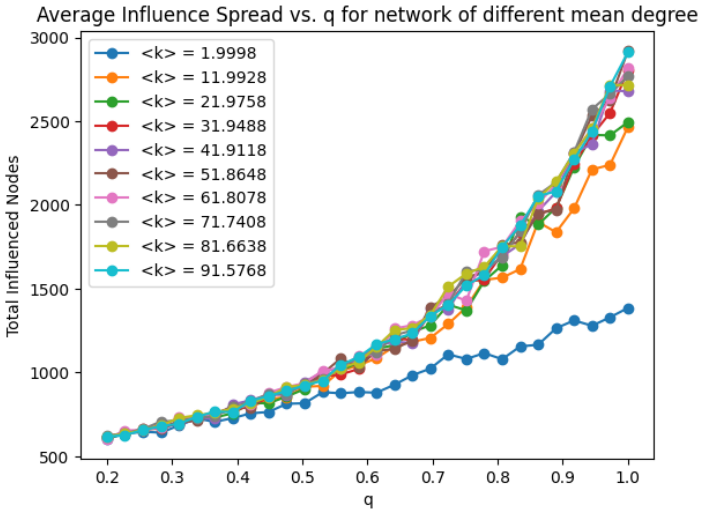

Figure 1: Influence spread dynamic is similar for values of $< k >> 1$

We keep $n = 10,000$ and $m = 10$ through the bulk of the investigation (when m is not the independent variable of the question). This is because for $m > 1$ the dynamics and outcome of the simulation on the network is independent of m as seen in figure 1, keeping m small will result in faster simulation too. We will average around 30 runs for a fixed set of parameters' simulation.

2.2 Independent Cascade Model

According to D'angelo et al., 2016, the ICM is an information diffusion model where the information flows over the network through cascade. The process runs in discrete steps.At the beginning of ICM process, few nodes are given the information known as seed nodes. Upon receiving the information these nodes become active. In each discrete step, an active node tries to influence one of its inactive neighbors. In spite of its success, the same node will never get another chance to activate the same inactive neighbor. The success depends on the propagation probability of their connection. Propagation Probability of a connection (edge) is the probability by which one can influence the other node.The process terminates when no further nodes became activated from inactive state.
The independent cascade steps in the code are done below.
Nodes have three states :

- Inactive (I) - Node unaware of information orYb not influenced
- Active (A) - Node already influenced by information in diffusion
- Actived (D) - Nodes activated other nodes at prev round, but can not activate other nodes anymore.

Then the cascade steps are :

1. Select a fraction, $0 < p_{seed} < 1$ as the initial active nodes, as seeds.
2. Calculate the influence probability p, of each half edge with

$$p_{ij} = \frac{q}{k_j} \tag{5}$$

   Where j is the target node and i is the active node. $0 < q <= 1$ is the influence threshold, a hyper parameter fixed for a particular simulation.

3. In each round, the propagation will be cascaded only once, and the active nodes in the round will be set to actived node, which would **not** cascade to other nodes any more.
4. Keep cascading until we have no more active nodes. The number of actived nodes are the total influenced nodes.

2.3 Seeding Protocols

Simlar to marketing/broadcasting in the real world. We can choose better seed that result in a greater number of total influenced nodes. Here are some selection policies we will consider in the report :

1. **Random** Select seeds randomly from all the nodes.
2. **Degree** Calculate the degree k_i of each node and select the $p_{seed} * n$ nodes of highest degree as seeds.
3. **Fixed degree** This addressed the issue with degree ranking. When we pick of node with high degree but most of its neighbours are actived, then these nodes will no longer be considered any more, so the effective degree of the node is actually lower. To fix this, after the selection of a seed, update the neighbours' (effective) degree with -1 to make sure those nodes do not consider the seed anymore.
4. **Neighbour degree** Select seed by nodes' centrality score (highest)

$$C_i = k_i + \sum_j k_j A_{ij} \tag{6}$$

5. **Fixed neighbour degree** Update the neighbours' centrality score with $-C_{seed}$.

The time taken to select the seeds will be obtained using Python's **'timeit'** module.

2.4 Measuring Impact through Effective Influence

We attempt to quantify the impact of seed selection by defining a quantity known as E, the effective influence

$$E = \frac{< D >}{p_{seed} * n} - 1 \tag{7}$$

Where $< D >$ is the mean influenced nodes at termination, the -1 accounts for the seed itself. Physically, the quantity represent the mean number of influence achieved per seed.

3 Results

3.1 Evolution of inactive, active and actived nodes over time

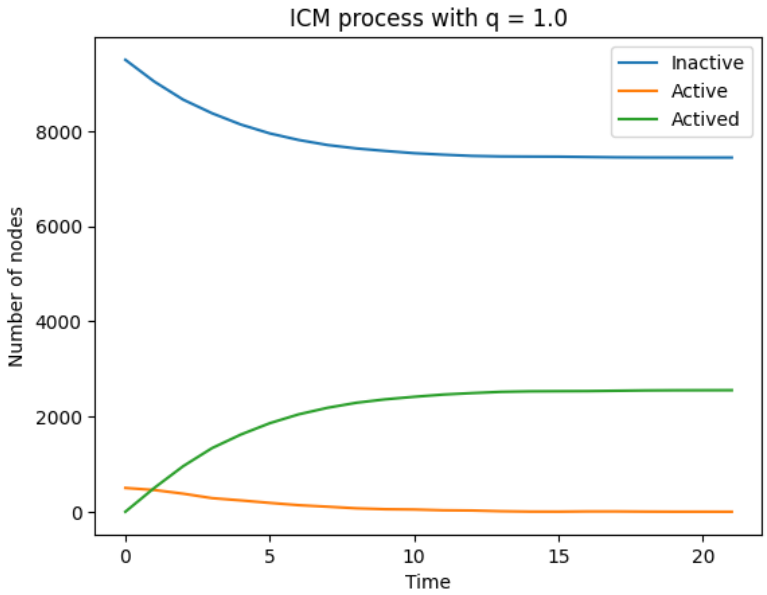

Figure 2: Number of actived nodes in ICM grows then plateaus over time.

3.2 Average influence spread vs. q for different seeding protocols

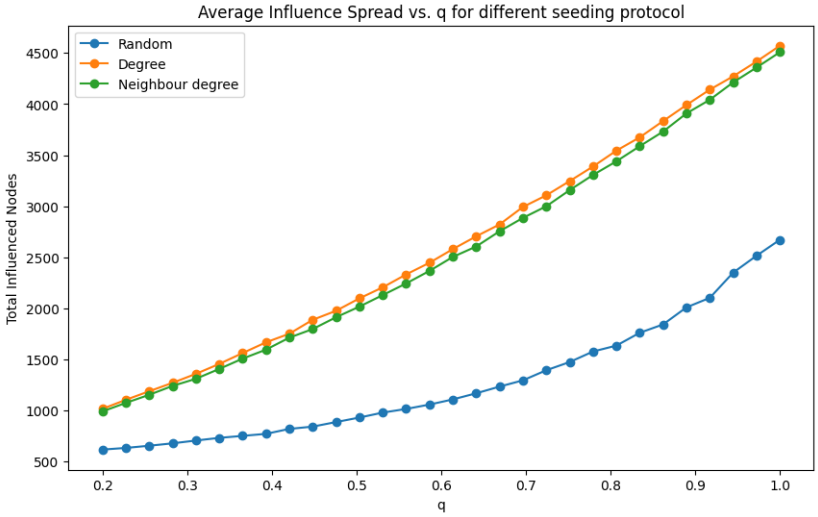

Figure 3: Influence spread for degree,neighbour degree seeding is much greater than random seeding for all values of q.

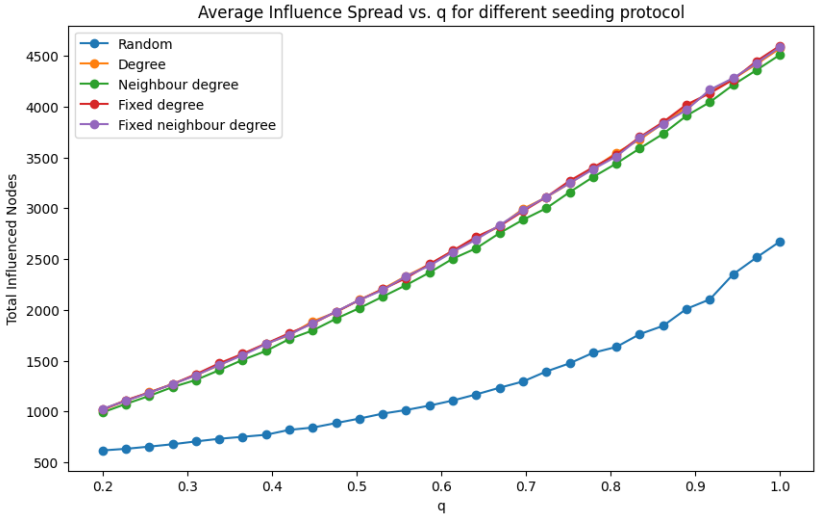

Figure 4: Fixed degree seedings do not show improvement in influence spread compared to degree seedings for all values of q.

3.3 Time taken to select seed for different seeding protocol

Protocol	**Time (s)**
Random	0.0269
Degree	0.124
Neighbor Degree	3.33
Fixed Degree	23.1
Fixed Neighbor Degree	37.5

Table 1: Average Time taken to select seeds for different seeding protocol in seconds, with $p_{seed} = 0.05$. In increasing order it is random,degree,fixed degree, neighbour degree then fixed neighbour degree.

3.4 Influence vs. seed size

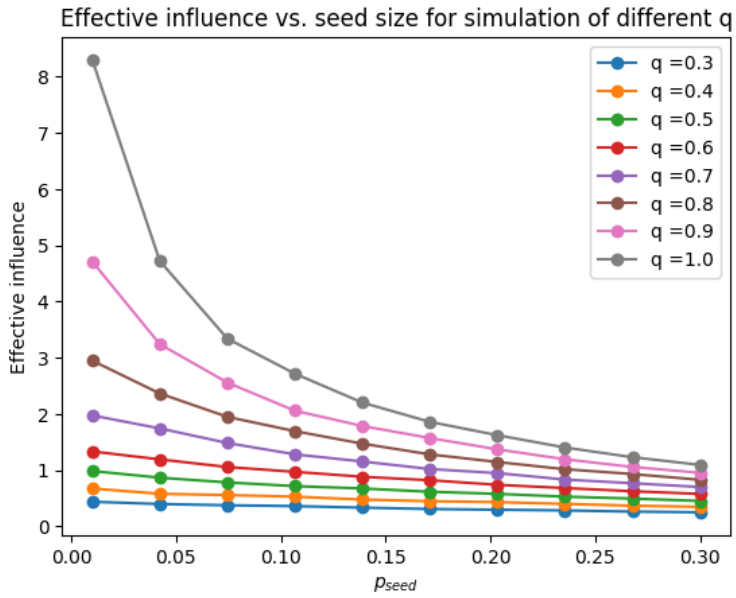

Figure 5: Effective influence and seed size has an approximately inversely proportional relationship for all values of q.

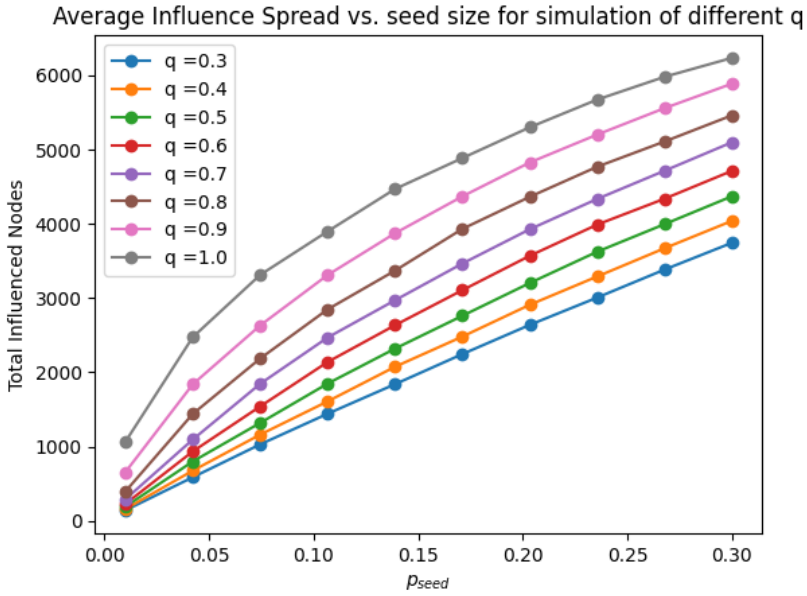

Figure 6: Influence spread and seed size shows a positive relationship for all values of q.

4 Discussion

4.1 Influence spread vs. q

Given that a node of degree k is activated, the expected number of node it will influence is

$$E(Y|k) = k \sum_{k'} P(k'|k) \frac{q}{k'} \tag{8}$$

Where $p(k'|k)$ is the probability of a node of degree k' being the neighbour of a node of degree k. In the Barabasi-Albert network this is given by

$$p(k'|k) = \frac{m(k+2)}{kk'(k'+1)}(1 - \frac{\binom{2m+2}{m+1}\binom{k+k'-2m}{k'-m}}{\binom{k+k'+2}{k'+1}} \tag{9}$$

For the random seeding protocol, the expected number of influence given an activated node is simply $E(Y|k)$ averaged over the degree distribution since the seeds are chosen uniformly at random :

$$\begin{aligned} E(Y|random) &= \sum_k E(Y|k)p(k) \\ &= \sum_k p(k)k \sum_{k'} p(k'|k)\frac{q}{k'} \end{aligned} \tag{10}$$

However, if the degree seeding protocol were chosen instead, the calculation of $E(Y)$ weighs (non-uniformly) $E(Y|k)$ with the probability of a node with degree (k) being activated, $\alpha(k)$, since we pick the nodes with highest degree as seeds.

$$\begin{aligned} E(Y|degree) &= \sum_k E(Y|k)p(k)\alpha(k) \\ &= \sum_k p(k)\alpha(k)k \sum_{k'} p(k'|k)\frac{q}{k'} \end{aligned} \tag{11}$$

In the Barabasi-Albert network, we can approximate the nearest-neighbor degree distribution of a node k with

$$p(k'|k) \approx k'p(k') \tag{12}$$

which gives

$$E(Y|k) \approx qk \tag{13}$$

In the degree seeding protocol, $\alpha(j) >= \alpha(i)$ for $j >= i$, firstly with how the seeds are chosen and secondly with the presence of friendship paradox in scale-free networks Marcos Amaku; Rafael I. Cipullo ; Jos´e H. H. Grisi-Filho and Ossada, 2014.
Therefore using 13 and the arguments from above, we know that $E(Y|degree) \geq E(Y|random)$.
Let us denote α as the mean probability of a node being influenced. The number of influenced nodes expected in the network, assuming it is large is

$$n\alpha = np_{seed} + np_{seed}E(Y) + np_{seed}E(Y)^2 + ... + np_{seed}E(Y)^D \tag{14}$$

Where D is the network diameter. Simplifying the expression gives

$$\alpha = p_{seed}\sum_{d=0}^{D} E(Y)^d \tag{15}$$

We expect

$$\alpha(degree) \geq \alpha(random)$$

Figure 3 demonstrates the drastic increase in $< D >$ from using degree seeding protocol vs. random seeding protocol. This matches the expectation based off the equations we have derived above. However, it is surprising that the neighbour degree protocol presented a slightly lower influence spread than degree seeding protocol.
According to figure 4, fixed degree had basically no improvement in influence spread compared to degree seeding protocol. Fixed neighbour degree seeding had a slight increase in influence spread from neighbour degree but was basically the same values as degree seeding.

4.2 Effective influence vs. seed size

Since the cost in marketing is correlated to how many initial broadcasting points (eg. number of influencers, brand ambassadors, newspaper company) the company invests in, it would have been meaningful to observe the relationship between effective influence vs. p_{seed} for different q. We observe from figure 5 that for all q values, effective influence has an inversely proportional relationship with p_{seed}, with the rate of change increasing as q increases. However, figure 6 does show that the final influence spread increases with p_{seed} for all q, this is expected from equation 15. So for value per person/node wise it is ideal to have a lower p_{seed} but final influence size wise p_{seed} should be increased. This suggests a trade-off relationship that can be investigated in future research.

5 Conclusion

It is difficult to derive and test the expressions relating $< D >$ to q,p_{seed} and network parameters analytically. However, simulations have shown us that for a particular network and q, the degree seeding protocol gave the best value in terms of improving average influence to time taken to select seeds, compared to fixed degree seed selections and neighbour degree seed selections. So is the best seeding protocol out of the ones we've investigated in this paper for marketing companies working with similar social networks.

6 Limitations and future research

More rigorous analytical analysis could be carried out to verify improvements from seeding protocols to seeding protocols, especially the reason why neighbour degree seeding underperformed compared to degree seeding. The usage of scale-free network in this investigation to mimic real-life influence spread is also limited to the fact that Broido; A.D. ; Clauset, 2019 found robust evidences that strongly scale-free network is empirically rare. For most network, log-normal distributions fit the data as well or better than power laws. Furthermore, social networks are at best weakly scale free, while a handful of technically or biological networks appear strongly scale free. Therefore such structural diversity of real-world networks suggest the need to repeat above investigation of networks with different degree distributions.
In modelling the impact of social media marketing on social network, it might be suggested sometimes that the Linear Threshold Model which captures richer dynamics is a better model. This is because followers are likely to purchase, click on a link etc. after repeated exposure or collective influence matters, like new product. Consider repeating above investigation with a LTM simulation instead.

References

BARABÁSI, A.-L. (n.d.). Network science : The barabási-albert model.

Barabási, R., Albert-László; Albert. (1999). Emergence of scaling in random networks. *Science*, *286*(5439), 509–512. https://doi.org/https://doi.org/10.1126%2Fscience.286.5439.509

Broido; A.D. ; Clauset, A. (2019). Scale-free networks are rare. *Nat Commun*, *10*(1017). https://doi.org/https://doi.org/10.1038/s41467-019-08746-5

D'angelo, G., Severini, L., & Velaj, Y. (2016). Influence maximization in the independent cascade model. *Italian Conference on Theoretical Computer Science*. https://api.semanticscholar.org/CorpusID:6774953

Lewis, J. (2023). *A study on the importance of social network analysis in modern social media marketing strategy including the impact of economic uncertainty.* https://medium.com/@josephlewis1402/a-study-on-the-importance-of-social-network-analysis-in-modern-social-media-marketing-strategy-c2b96e392432

Marcos Amaku; Rafael I. Cipullo ; Jos´e H. H. Grisi-Filho, F. S. M., & Ossada, R. (2014). The friendship paradox in scale-free networks. *Applied Mathematical Sciences*, *8*(37), 1837–1845. https://doi.org/http://dx.doi.org/10.12988/ams.2014.4288

Molak, A. (2017). *A step-by-step barabási–albert model in python 3.* https://github.com/AlxndrMlk/Barabasi-Albert_Network/blob/master/README.md

Scale-free networks. (n.d.). https://mathinsight.org/scale_free_network

Shakarian, P., Bhatnagar, A., Aleali, A., Shaabani, E., & Guo, R. (2015, January). The independent cascade and linear threshold models. https://doi.org/10.1007/978-3-319-23105-1_4

A Python code to import modules used

```
import numpy as np
from matplotlib import pyplot as plt
from timeit import timeit
import networkx as nx
```

B Python code to generate BA Network and run ICM simulation

B.1 ICM Simulation class

```
class ICM_Model():
    """ I: Inactive, A:Active, D:Actived"""
    def __init__(self, network : nx.Graph, q, initial_seeds = None ,
        p_seed=0.05) -> None:
        self.network = network
        self.q = q
        self.p_seed = p_seed

        # I,A,D states
        self.I = {n for n in self.network}
        self.A = set()
        self.D = set()

        # Initially activate a small fraction of the population
        if not initial_seeds:
            initial_seeds = np.random.choice(list(self.I),
            size=int(self.p_seed*self.network.number_of_nodes()),
            replace=False)

        self.A.update(initial_seeds)
        self.I.difference_update(self.A)

    def run(self):
        '''Runs simulation for a cycle'''

        next_A = set()
        for node in self.A:
            for j in self.network.neighbors(node):
                if j in self.I and np.random.rand() <
                (self.q/self.network.degree(j)): # type: ignore
                    next_A.add(j)

        self.D.update(self.A)
        self.A = next_A
        self.I.difference_update(next_A)

    def run_to_extinction(self) -> tuple[list[int],list[int],list[int]]:
        '''Run simulation until no more active node is left, then
        returns time series of IAD numbers as arrays.'''

        I_list, A_list, D_list = [len(self.I)],[len(self.A)],[len(self.D)]

        while self.A:
            self.run()

            I_list.append(len(self.I))
```

```
            A_list.append(len(self.A))
            D_list.append(len(self.D))

        return I_list, A_list, D_list
```

B.2 Generate Network and view time series of I,A,D nodes

```
n = 10000
m = 10
G = nx.barabasi_albert_graph(n,m)
q=0.2
simulation = ICM_Model(G,q,p_seed=p_seed)
I,A,D = simulation.run_to_extinction()

plt.plot(I, label='Inactive')
plt.plot(A, label='Active')
plt.plot(D, label='Actived')
plt.xlabel('Time')
plt.ylabel('Number of nodes')
plt.title('ICM process with q = {}'.format(q))
plt.legend()
plt.show()
```

C Python code to select seeds with different seeding protocol

C.1 Method to compute average influence spread for a set of seeds and array of q

```
def average_D_vs_q(G, q_array, initial_seeds=None, p_seed=0.05, runs=30):
    avg_D = []

    for q in q_array:
        D_values = []
        for _ in range(runs):
            simulation = ICM_Model(G,q,initial_seeds,p_seed)
            I,A,D = simulation.run_to_extinction()
            D_values.append(D[-1])
        avg_D.append(np.mean(D_values))

    return avg_D
```

C.2 Random seeding

Just let parameter `initial_seeds` of method `average_D_vs_q` be None.

```
p_seed=0.05
q_array = np.linspace(0.2,1.0,30)
avg_D = average_D_vs_q(G, q_array,p_seed=p_seed)
```

C.3 Degree Seeding

```
def degree_seeding(G, num_seeds):
    nodes_degree_sorted = sorted(G.degree(), key=lambda x:x[1],
    reverse=True)
    return [n for n,d in nodes_degree_sorted[:num_seeds]]
```

```
deg_seeds = degree_seeding(G, int(n*p_seed))
avg_D_deg = average_D_vs_q(G, q_array, deg_seeds)
```

C.4 Neighbor degree seeding

```
def neighbour_degree_seeding(G, num_seeds):
    centrality = {}
    for node in G:
        centrality[node] = G.degree[node]
        for j in G.neighbors(node):
            centrality[node] += G.degree[j]

    nodes_centrality_sorted = sorted(centrality.items(),
                        key=lambda x:x[1], reverse=True)
    return [n for n,d in nodes_centrality_sorted[:num_seeds]]

neigh_deg_seeds = neighbour_degree_seeding(G, int(n*p_seed))
avg_D_neigh_deg = average_D_vs_q(G, q_array, neigh_deg_seeds)
```

C.5 Fixed degree seeding

```
def fixed_degree_seeding(G, num_seeds):

    seeds = []
    degree = list(G.degree())

    for _ in range(num_seeds):
         nodes_degree_sorted = sorted(degree, key=lambda x:x[1], reverse=True)
         curr = nodes_degree_sorted[0][0]
         seeds.append(curr)
         degree[curr] = (curr,-1)
         for j in G.neighbors(curr):
              degree[j]= (j, degree[j][1]-1)

    return seeds
fixed_deg_seeds = fixed_degree_seeding(G, int(n*p_seed))
avg_D_fixed_deg = average_D_vs_q(G, q_array, fixed_deg_seeds)
```

C.6 Fixed neighbor degree seeding

```
def fixed_neighbour_degree_seeding(G, num_seeds):

    seeds = []
    centrality = {}
    for node in G:
        centrality[node] = G.degree[node]
        for j in G.neighbors(node):
            centrality[node] += G.degree[j]

    for _ in range(num_seeds):
         nodes_centrality_sorted = sorted(centrality.items(),
         key=lambda x:x[1], reverse=True)
         curr = nodes_centrality_sorted[0][0]
         seeds.append(curr)
         centrality[curr]=-1
```

```
            for j in G.neighbors(curr):
                centrality[j]-=G.degree[curr]

    return seeds
fixed_neigh_deg_seeds = fixed_neighbour_degree_seeding(G, int(n*p_seed))
avg_D_fixed_neigh_deg = average_D_vs_q(G, q_array, fixed_neigh_deg_seeds)
```

D Python code to time seed selection of different seeding protocol

```
SETUP_CODE="""
from __main__ import degree_seeding
from __main__ import neighbour_degree_seeding
from __main__ import fixed_degree_seeding
from __main__ import fixed_neighbour_degree_seeding
import networkx as nx
import numpy as np
n=10000
m=10
p_seed=0.05
q_array = np.linspace(0.2,1.0,30)
G = nx.barabasi_albert_graph(n,m)"""

time_rand = timeit(stmt='np.random.choice(list({n for n in G}),
                    size=int(p_seed*G.number_of_nodes()), replace=False)',
                    setup=SETUP_CODE, number=20)
time_deg = timeit(stmt='degree_seeding(G, int(n*p_seed))',
                  setup=SETUP_CODE, number=20)
time_neigh_deg = timeit(stmt='neighbour_degree_seeding(G, int(n*p_seed))',
                        setup=SETUP_CODE, number=20)
time_fixed_deg = timeit(stmt='fixed_degree_seeding(G, int(n*p_seed))',
                        setup=SETUP_CODE, number=20)
time_fixed_neigh_deg = timeit(stmt='fixed_neighbour_degree_seeding(G,
                        int(n*p_seed))', setup=SETUP_CODE, number=20)
```

E Python code to investigate influence spread as a function of seed size

E.1 Average influence

```
def average_D_vs_p_seed(G, p_seed_array,q,initial_seeds=None,runs=20):
    avg_D = []

    for p_seed in p_seed_array:
        D_values = []
        for _ in range(runs):
            simulation = ICM_Model(G,q,initial_seeds,p_seed)
            I,A,D = simulation.run_to_extinction()
            D_values.append(D[-1])
        avg_D.append(np.mean(D_values))

    return avg_D

q_array = [0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
```

```
p_seed_array = np.linspace(0.01,0.30,10)
n = 10000
m = 10
G = nx.barabasi_albert_graph(n,m)
for q in q_array:
        plt.plot(p_seed_array, average_D_vs_p_seed(G, p_seed_array, q),
                marker='o', label='q ={}'.format(q))

plt.xlabel(r'$p_{seed}$')
plt.ylabel('Total Influenced Nodes')
plt.title('Average Influence Spread vs. seed size
            for simulation of different q')
plt.legend()
plt.show()
```

E.2 Effective influence

```
for q in q_array:
    plt.plot(p_seed_array, np.array(average_D_vs_p_seed(G, p_seed_array, q))
            /(n*np.array(p_seed_array))-1, marker='o', label='q ={}'.format(q))

plt.xlabel(r'$p_{seed}$')
plt.ylabel('Effective influence')
plt.title('Effective influence vs. seed size for simulation of different q')
plt.legend()
plt.show()
```