Analysis of Facebook Network Structure and Spread of Information

Team 201:
Jeff Lim, Humphrey George Chua, Ronnie Lai, Toh Heng Wai, Toh Hong Kai, Toh Ker Yang

November 2018

Contents

1 Abstract

The objective of this project is to find a network structure that most closely resembles the network structure of Facebook. Facebook data was downloaded from Stanford Large Network Dataset Collection. We first summarized the 4 random network models introduced in the course (Erdős–Rényi model, Watts-Strogatz model, Barabási–Albert model, Mediation-Driven Attachment model) and evaluated how well they resemble the network structure of Facebook by comparing their network properties. We then created our own random network model in an attempt to further mimic the network structure of Facebook. Lastly, we simulated the spread of information over the best network model using the Susceptible - Infectious - Recovered (SIR) Model and compared it with real world data.

2 Network Properties

A network is a system of interconnected people, in this case their Facebook accounts. This can be represented as a graph, with each vertex symbolizing a Facebook account, and two Facebook accounts are connected if there exists an edge between them. In order to mimic the Facebook network, we need to look at a few properties of graphs and the way they affect the spread of information.

2.1 Average Vertex Degree

The vertex degree refers to the number of vertices a vertex is connected to. The average vertex degree can be calculated by the formula $\frac{2|E|}{|V|}$, where $|\ E\ |$ denotes the number of edges and $|\ V\ |$ denotes the number of vertices. Having a higher average vertex degree means that information from a single vertex can be spread to more neighbours. This allows for more rapid and efficient information spread in the network.

2.2 Average Path Length

The distance between two vertices is the length of the shortest path between them. The average path length is the average of the distances between all pairs of vertices. This gives us the number of edges information passes through on average to get from one vertex of a network to another. Intuitively, having a shorter average path length is advantageous for spread of information in a network.

2.3 Diameter

The diameter of a network is the maximum eccentricity of its vertices, where the eccentricity of a vertex is the maximum distance a vertex has from all other vertices in the network. Having a smaller diameter value thus improves the spread of information as information from one vertex needs to pass through less edges in order to reach other vertices of the network.

2.4 Number of Triangles

A triangle in a network refers to a set of three vertices that are all connected to each other. This is representative of what we know in the real world as 'mutual friends'. In such a set, the average vertex degree is maximized. and the average path length is minimized. Having more triangles occurring in a network thus means the network is more connected in path length of 1.

2.5 Clustering Coefficient

The clustering coefficient(CC) measures the degree to which vertices are clustered together. The CC of a vertex is calculated as the number of triangles containing said vertex divided by the total number of triangles that can potentially be formed between the vertex and its neighbours. Thus, the average CC is closely related to the total number of triangles in a network for the numerators, and the distribution of vertex degrees for the numerators. Relating this to the spread of information, information is more easily spread in a network with a higher average CC since there are more triangles.

3 Facebook Network

A set of data that shows part of the complex networking of Facebook was downloaded from Stanford Large Network Dataset Collection [1] and plotted as shown below.

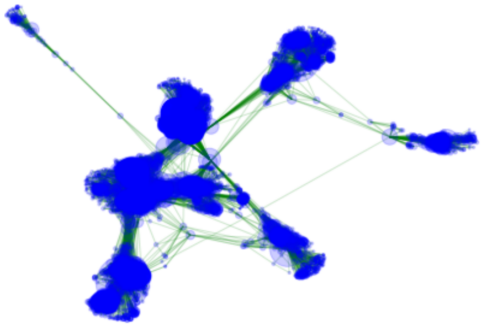

Figure 1: Facebook Network Graph

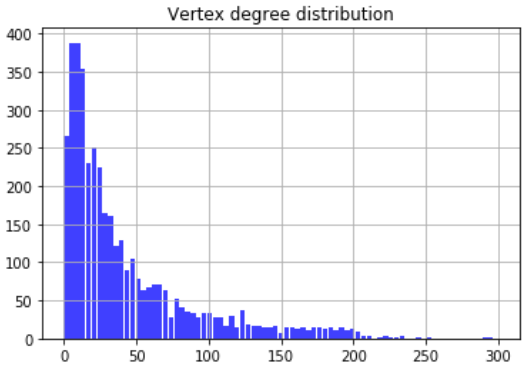

Figure 2: Distribution of Vertex Degrees

FB network	Values
$\mid V \mid$	4039
$\mid E \mid$	88234
CC	0.6055
Avg. Path Length	3.6925
Avg. Vertex Degree	43.691
No. of Triangles	1612010
Diameter	8

Table 1: Properties of Facebook(FB) Network

In the real world, people tend to form cliques, represented by separation of the network into multiple clusters. Within these cliques, people also tend to have many mutual acquaintances between one another. This explains the large number of triangles and the high clustering coefficient of 0.6055. The small average path length of 3.7 coincides with the global phenomenon of six degrees of separation [2], which is where the average distance between any two people is counter-intuitively small. Lastly, a large average vertex degree of 43.7 reflects the fact that in the real world, each person is connected to many other people. In the next section, we will be looking at different models as candidates for mimicking our Facebook network.

4 Models

4.1 Erdős–Rényi(ER) Model

The Erdős–Rényi(ER) model is a random network model constructed with two parameters n and p. n refers to the number of vertices in the network and each pair of edges is connected with probability p. As a result, with a higher probability p, more edges are likely to be formed between vertices, forming a more densely connected random network. Note that each pair of vertices are linked independently with probability p. As a result, the distribution of vertex degrees of ER model follows a binomial distribution.

To construct a random graph of this model with reference to the Facebook data, the parameter n is given as 4039. Meanwhile, the parameter p is given by the following formula:

$$p = \frac{number\ of\ actual\ edges}{maximum\ possible\ number\ of\ edges} = \frac{88234}{C(4039,2)} = \frac{88234}{4039 \times 4038 \times 0.5} = 0.01081$$

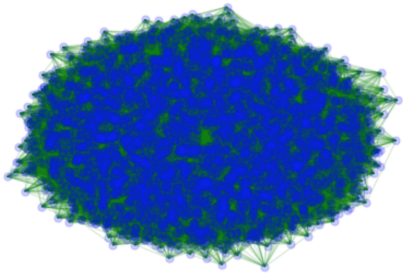

Figure 3: Erdős–Rényi Network Graph with parameters $n = 4039,\ p = 0.01081$

Model	$\mid V \mid$	$\mid E \mid$	CC	Avg. Path Length	Avg. Vertex Degree	No. of Triangles	Diameter
FB	4039	88234	0.6055	3.6925	43.691	1612010	8
ER	4039	88574	0.0109	2.6034	43.859	14128	4

Table 2: Comparison between Facebook(FB) ER networks

Based on the results generated above, we observed that the number of edges formed in the ER model is similar to that of the Facebook model. The small difference is because the number of edges generated with the ER model is random. More importantly, observe that there is a huge difference between their average clustering coefficients, with Facebook having a high CC of 0.6055 and the ER network having a low CC of 0.0107.

By comparing the properties of the ER network and Facebook network, we can observe that the differences in the diameters, average path length, number of triangles are all significant. Only the average degrees are similar. Hence, we can conclude that the ER model is not suitable for mimicking the network structure of Facebook.

4.2 Watts–Strogatz(WS) Model

The Watts–Strogatz(WS) model is a random network model characterized by small-world properties such as short average path length and high clustering coefficient. The small-world phenomenon refers to the case that in many real-world networks, the standard distance between vertices remains unusually small despite the network growing very large. This principle was introduced by Stanley Milgram in the 1960's. [2]

The WS network is constructed with three parameters n, k and p. n refers to the number of vertices in the network. A ring lattice with these vertices is first formed, where each vertex is connected to the k nearest vertices($\frac{k}{2}$ neighbours on each side). Then, each edge is rewired with probability p to a different vertex, chosen from all possible vertices uniformly while avoiding self-loops and edge duplication.

To construct a random graph of this model with reference to the Facebook data, the parameter n is given as 4039. the parameter k is the calculated by the closest even number to $2\frac{|E|}{|V|} \approx 44$. Lastly, the parameter p is calculated from the formula $\frac{3(k-2)}{4(k-1)}(1-p)^3 = 0.6055$, which gives $p = 0.0536$.

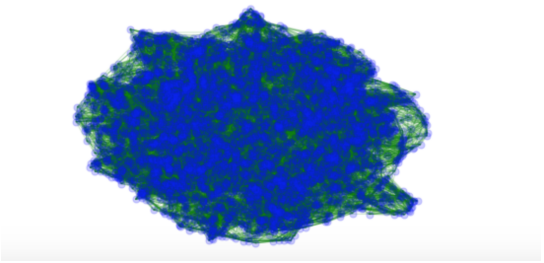

Figure 4: Watts–Strogatz Network Graph with parameters $n = 4039,\ k = 44,\ p = 0.0536$

Model	$\mid V \mid$	$\mid E \mid$	CC	Avg. Path Length	Avg. Vertex Degree	No. of Triangles	Diameter
FB	4039	88234	0.6055	3.6925	43.691	1612010	8
WS	4039	88858	0.6055	3.2184	44.000	800625	5

Table 3: Comparison between Facebook(FB) and WS networks

Compared to the ER model, the WS model is a better model for the real-world network due to its high clustering coefficient of 0.6055, which matches that of the Facebook network. The average path length and diameter of the WS network is also closer to that of the Facebook network. However, the number of triangles is curiously lower than that of the Facebook network. Since the clustering coefficients are the same, the difference is likely due to the distribution of vertex degrees. Furthermore, the shape of the graph is too dissimilar to that of the Facebook network. Therefore we can conclude that the WS model is not good enough to mimic the Facebook network.

4.3 Barabási–Albert(BA) Model

The Barabàsi–Albert(BA) model is an algorithm for generating random scale-free networks using a preferential attachment mechanism. Scale-free implies that the vertices in network exhibit a power-law distribution. This is similar to how connections are formed in Facebook; if a user has more friends, the user will have a higher chance of making new friends.

The BA model has two important features: growth and preferential attachment. Growth refers to the increasing in number of vertices over time, while preferential attachment means that if a vertex has more edges, it has a higher probability of receiving more new edges.

The BA network is constructed with two parameters n and m, where n is the number of vertices and m is the number of vertices each new vertex is to be connected to, and $n > m$. The graph is initialized as a star graph with a single vertex connected to m other vertices. Vertices are then added one by one, each one connecting to m pre-existing vertices, until there are n vertices. Adding each edge individually and avoiding repeated edges, the probability of a vertex i receiving a new edge, denoted p_i, follows the formula:

$$p_i = \frac{k_i}{\sum_{j=1}^{N} k_j}$$

where k_i is the degree of vertex i and N is the number of existing available vertices. Thus, to construct a BA network with reference to the Facebook network, the parameter n is given as 4039, and we calculate m to be the nearest integer to half of the average vertex degree, so $m = 22$.

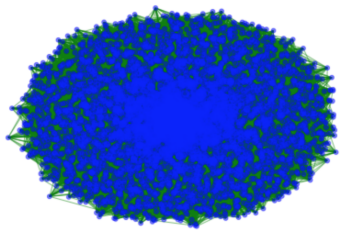

Figure 5: Barabási–Albert Network Graph with parameters $n = 4039,\ m = 22$

Model	$\mid V \mid$	$\mid E \mid$	CC	Avg. Path Length	Avg. Vertex Degree	No. of Triangles	Diameter
FB	4039	88234	0.6055	3.6925	43.691	1612010	8
BA	4039	88374	0.0375	2.5123	43.760	86773	4

Table 4: Comparison between Facebook(FB) and BA networks

We can observe from the comparisons that the average clustering coefficient and the number of triangles are significantly lower at 0.0375 and 86773 respectively. Also, the BA network has a significantly shorter average path length and its diameter is half that of the Facebook network. Therefore, the BA model is not able to mimic the Facebook network well, just that the rate of spread of information for the BA model might not be a good representation for the Facebook model.

4.4 Mediation-Driven Attachment(MDA) Model

The Mediation Driven Attachment(MDA) model is similar to the BA model. It uses the same parameters n and m. Each new vertex chooses an existing vertex at random as a mediator. Then the new vertex connects not with the mediator but with m of mediator's neighbors randomly. Like the BA model, the MDA model uses a preferential attachment mechanism since an already well-connected vertex has more mediators through which it can gain even more connections.[3]

Suppose there are N existing vertices and consider the vertex i with degree k_i. This implies that it has k_i neighbours. Assuming that its neighbours have degree $k_1, k_2, \ldots, k_{k_i}$, the probability of a vertex i receiving a new edge, denoted p_i, follows the formula:

$$p_i = \frac{\sum_{j=1}^{ki} \frac{1}{k_j}}{N}$$

To construct the MDA network, we can choose the same parameters as we did for the BA network, with $n = 4039$ and $m = 22$.

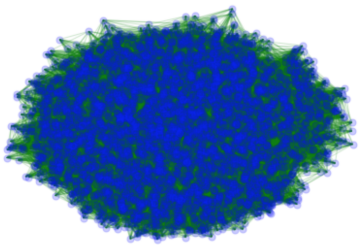

Figure 6: Mediation-Driven Attachment Network Graph with parameters $n = 4039,\ m = 22$

Model	$\mid V \mid$	$\mid E \mid$	CC	Avg. Path Length	Avg. Vertex Degree	No. of Triangles	Diameter
Facebook	4039	88234	0.6055	3.6925	43.691	1612010	8
BA	4039	88374	0.0375	2.5123	43.760	86773	4
MDA	4039	88605	0.0439	2.5098	43.874	105243	4

Table 5: Comparison between Facebook(FB), BA and MDA networks

Based on the results generated above, we observed that the properties and shape of the MDA network is similar to the BA network, albeit with a slightly higher average CC and a larger number of triangles. Nevertheless, the MDA network, like the BA network, is not suitable for mimicking the Facebook network.

4.5 Custom Random Network Model 201

In order to create a random network model that closely resembles the Facebook network, we reverse engineered the network and integrated the parts of different models that replicate each characteristic of the Facebook network. We have noticed that the model is split into many clusters of different sizes connected via a few edges between each cluster, a feature not exhibited in any of the previous models. Noting that there is a high average CC, we deduced that each cluster itself has to have a high average CC. Also, larger clusters appear to be more connected to each other than smaller clusters.

This motivates a 2-step model. We start with a cluster formation step, where WS networks are used to form each cluster in order to obtain a high average CC. The clusters are sorted along the diagonal of the adjacency matrix to have the largest clusters in the middle and the smallest clusters at the ends to create a 'distance' between each cluster. Next, we move on to the cluster connection step. The clusters are connected to each other by applying the ER model concept to each combination of two clusters. The probability used for each pair of clusters is inversely proportional to the distance between them. This is to simulate a 'power law' distribution similar to the BA model where larger clusters become even more connected.

We will call this new model the Clustom Random Network Model(a play on words combining 'cluster' and 'custom'). The function for constructing a Clustom network takes in 5 parameters:

n the total number of vertices

g the number of clusters

l since each cluster is a WS network with a different n, the k value must vary accordingly. l is thus a number between 0 and 1 to control the k values using the following formula: $k = 2\left\lfloor \frac{nl^{\lceil \frac{n}{500} \rceil}}{2} \right\rfloor$

p the probability of edges being rewired in each WS graph

q the probability of edges being formed between clusters at distance 1

Due to the random distribution of vertices into the different clusters, the total number of edges between clusters varies greatly each time. This causes two flaws in the model: unpredictability of the total number of edges and a possibility that some clusters are not connected to the graph, causing the diameter and average path length to be uncountable. To prevent this, we had to use a while loop on the model to find a Clustom network that is connected and has a similar number of edges to the Facebook network for clear comparisons.

For the parameters n and g, we referred to the Facebook network to obtain $n = 4039$ and $g = 7$. It is a difficult problem to calculate optimum numbers for the other parameters. However, we found that adjusting them caused noticeable changes in the number of edges and CC. An increase in l increases the number of edges and CC since l affects the k values in the WS networks. An increase in p decreases the CC since the WS networks deviate more from being ring lattices. Lastly, an increase in q increases the number of edges and decreases the CC slightly since edges formed in the cluster connection step are not likely to form triangles, thus reducing the ratio. With this, we could use gradually adjust the parameters to achieve the desired expected number of edges. The parameters we found to give the best results were $l = 0.25$, $p = 0.09$ and $q = 0.0002$.

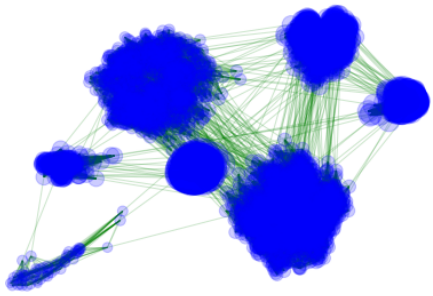

Figure 7: Clustom Network Graph with parameters
$n = 4039,\ g = 7,\ l = 0.25,\ p = 0.09,\ q = 0.0001$

Model	$\mid V \mid$	$\mid E \mid$	CC	Avg. Path Length	Avg. Vertex Degree	No. of Triangles	Diameter
Facebook	4039	88234	0.6055	3.6925	43.691	1612010	8
Clustom	4039	87310	0.5514	4.0806	43.308	1136273	9

Table 6: Comparison between Facebook and Clustom networks

Looking at the shape of the graph, it is very clear that the network shares the 'multiple cluster' feature with the Facebook network. The average CC and number of triangles is relatively high compared to the other networks, at 0.5514 and 1136273 respectively. The average vertex degree, average path length, and diameter are very similar to that of the Facebook network. We can thus conclude that the Clustom Random Network Model is able to mimic the Facebook network reasonably well.

5 Comparison of Models

5.1 Degree Distribution

Graphs are commonly compared via the distribution of the degrees of their vertices. Two networks with a similar vertex degree distributions are likely to share common traits with each other. Below are the vertex degree distributions of each of the networks. The x-axis of each histogram represents degree, and the y-axis represents the number of vertices.

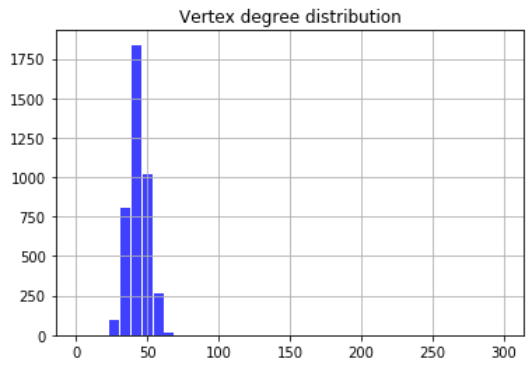

Figure 8: Erdős–Rényi Network Distribution of Vertex Degrees

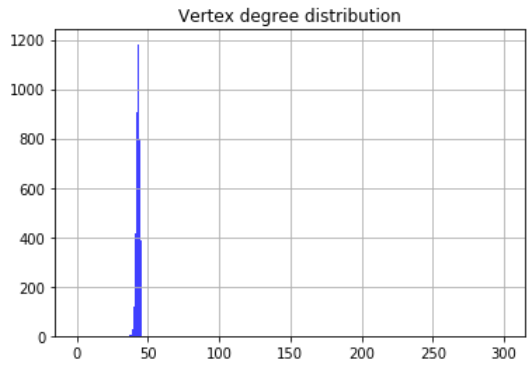

Figure 9: Watts–Strogatz Network Distribution of Vertex Degrees

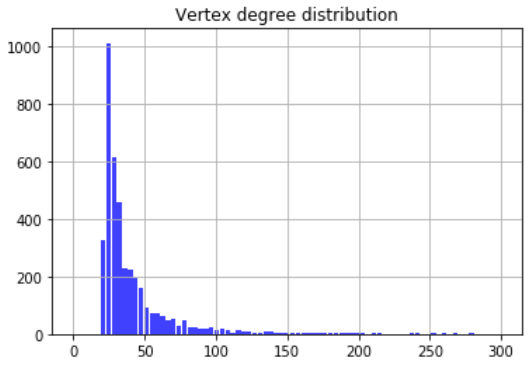

Figure 10: Barabási–Albert Network Distribution of Vertex Degrees

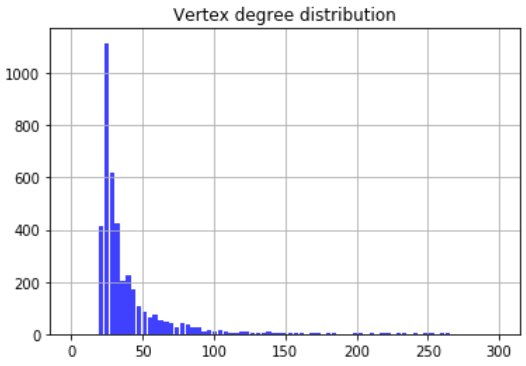

Figure 11: Mediation-Driven Attachment Network Distribution of Vertex Degrees

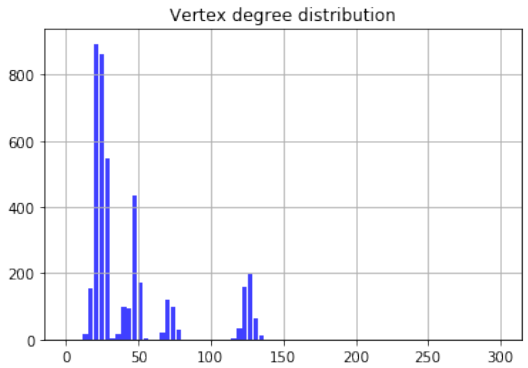

Figure 12: Clustom Random Network Distribution of Vertex Degrees

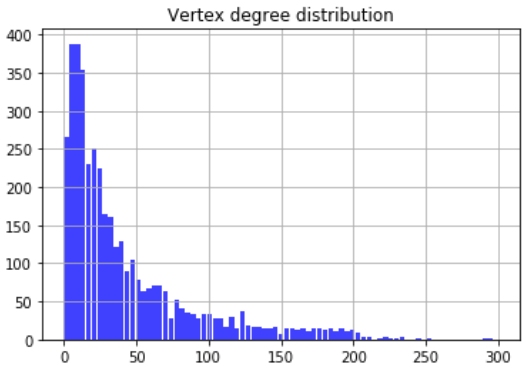

Figure 13: Facebook Network Distribution of Vertex Degree

As can be seen from the above histograms, the shape of the vertex degree distributions of the ER and WS networks are bell-shaped and relatively symmetric about their average vertex degrees. However, the vertex degree distribution in the Facebook Network is skewed right. Therefore, ER and WS networks would not be good representations of a real world network as they clearly show a different type of distribution of vertex degrees from the Facebook network.

On the other hand, the shape of vertex degree distributions of the BA and MDA networks are skewed right like in the Facebook network. Therefore, looking at the distribution of vetex degrees alone, BA and MDA networks are better candidates for mimicking the Facebook network.

As for the Clustom random network, the vertex degree distribution has multiple peaks, with bell-shapes relatively symmetrical about each peak, attributed to the fact that the Clustom random network is made up of multiple WS networks. However, looking at the histogram as a whole, we can see that the overall the shape of vertex degree distribution is skewed right like in the Facebook network. We speculate that with larger values of the number of vertices and clusters, the shape of the vertex degree distribution would likely tend towards that of the Facebook network.

5.2 Mimicking the Facebook Network

The ER model, WS model, BA model and MDA model each have some properties that are observed in the Facebook Network. Below is a table of the network properties of all of the networks for a side-by-side comparison. The networks are arranged in order of similarity to the Facebook Network.

Model	$\mid V \mid$	$\mid E \mid$	CC	Avg. Path Length	Avg. Vertex Degree	No. of Triangles	Diameter
FB	4039	88234	0.6055	3.6925	43.691	1612010	8
Clustom	4039	87310	0.5514	4.0806	43.308	1136273	9
WS	4039	88858	0.6055	3.2184	44.000	800625	5
MDA	4039	88605	0.0439	2.5098	43.874	105243	4
BA	4039	88374	0.0375	2.5123	43.760	86773	4
ER	4039	88574	0.0109	2.6034	43.859	14128	4

Table 7: Summary of Network Properties of each Network

Creating the Clustom Random Network Model put the parts that produce the desired properties from different models together. We generated a high average clustering coefficient using the WS Model, and put together a few of such networks using the concept of the ER model to create the separated clusters as seen in the Facebook Network. Using probability values inversely proportional to the distance between each pair of clusters, we were able to create a vertex degree distribution close to an exponential one. As a result, it is no surprise that the network properties of our Clustom model best mimics the Facebook network, and is the best candidate for investigating the spread of information, which we will talk about in the next section.

6 Spread of Information

6.1 SIR Model

The spread of information over a network can be modelled in a similar manner to the spread of epidemic diseases in the real world. Therefore, we are interested in using the Susceptible-Infected-Recovered(SIR) Model to test if the Clustom network indeed mimics real world networks. In the SIR model, each vertex has one of three statuses:

Susceptible: The vertex has yet to acquire the information but has the potential to be exposed to it and become infected as it is in the network.

Infected: The vertex has acquired information from the Facebook network and has the ability to spread it to its neighbouring vertices.

Recovered: The vertex has already received the information and subsequently lost all interest in anything regarding the information. Recovered vertices cannot be infected again.

In the first iteration, a select few vertices are infected while the rest are susceptible. Each iteration from then on starts with each vertex recovering with a fixed probability q. Then, each of the infected vertices has a chance to spread the information to its susceptible neighbours, should there be any, with a fixed probability p. The probability of a susceptible vertex getting infected is thus given by the formula $1-(1-p)^k$ where k is the total number of infected neighbours.

To conduct the SIR experiment on our Clustom Random Network Model, we first scaled down the total number of vertices to shorten the running time. We chose to scale down by a factor of ≈ 10 to make $n = 400$, which is not too small, so that the properties of the network are not compromised. Note that the parameters g, l *and* p remain constant due to their effect on the number of edges and the CC of each cluster. q has to be increased to $q = 0.001$ as the total number of edges between clusters has effectively been scaled down by a factor of ≈ 100, so we have to scale q up by a factor of 10. The resultant Clustom network below was used in our SIR experiment.

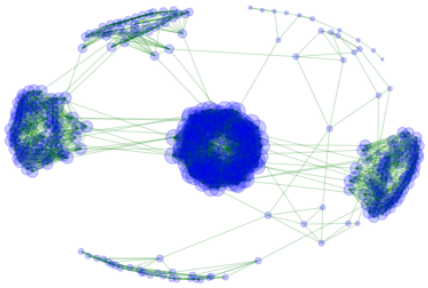

Figure 14: Clustom Network Graph with parameters
$n = 4039,\ g = 7,\ l = 0.27,\ p = 0.09,\ q = 0.001$

6.2 Google Trend (Trump-Kim Summit)

The news trend we have selected is the historical Trump-Kim Summit, which took place in Singapore on 12 June 2018. We extracted data from Google Trends [4] over 14 weeks from 20 May 2018 to 11 August 2018. Each value is taken to represent the number of infected nodes each week.

Week	0	1	2	3	4	5	6	7	8	9	10	11
Google Trend	3	5	14	100	11	3	6	3	3	0	3	0

Table 8: Comparison of SIR Trend with Google Trend (difference is 0.4106)

Note that the values for the Google trend are represented as a percentage of the highest number of searches reached in a week. To match this with our SIR, we need to convert the number of infected vertices to a percentage of the largest number of infected vertices reached in an iteration. We then try to match the initial number of infected nodes to be the same as the week 0 value in the Google trend. Assuming only about two-thirds of the population get infected in total, we calculate the initial number of infected vertices to be $\frac{2}{3} \times 3\% \times 400 = 8$. The SIR experiment will be run over 12 iterations.

6.3 Particle Swarm Optimization(PSO)

Particle Swarm Optimization(PSO) is a special algorithm for finding the minimum value of an objective function that does not have an explicit form. It is a computational method that optimizes a problem by trying to improve a candidate solution over many iterations. It solves a problem by having a population of candidate solutions, dubbed particles, and moving these particles around in the search-space around the particle's position based on its velocity. Each particle's movement is influenced by its local best known position, but is also guided toward the best known positions in the global search-space. These best positions will be updated as better positions are found by other particles as time passes. The process thus effectively moves the swarm towards the best solutions and will be terminated once the termination criterion is met, usually when the maximum number of iterations has been reached. [5]

In the case of our experiment, our objective is to efficiently and accurately model the spread of information about the "Trump-Kim Summit" using our Clustom network. This is to say we want to minimise the total deviation in the values between the modelled spread of information and the actual spread of information from the Google trend, calculated by the following formula:

$$deviation = \sqrt{\sum (true\ value - estimated\ value)^2}$$

This can be achieved by applying the PSO function to search for the optimal values of p and q which over the points in the unit square $\{(x, y) : 0 < x < 1,\ 0 < y < 1\}$. Because the SIR trend is different each time even for the same p and q values, we fix the initial infected vertices using systematic sampling to limit the randomness of the results. After which we plot the values of the Google trend and the optimum SIR trend to compare them.

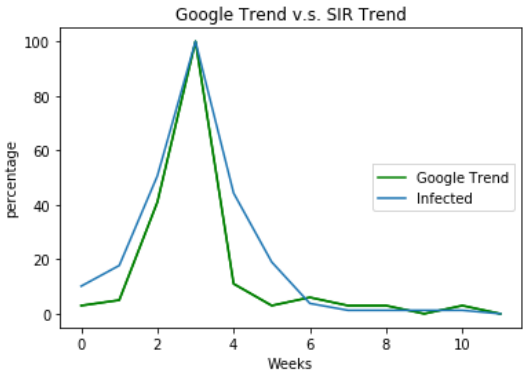

Figure 15: Google Trend v.s. SIR Trend

Week	0	1	2	3	4	5	6	7	8	9	10	11
Google Trend	3	5	14	100	11	3	6	3	3	0	3	0
SIR Trend	10.13	17.72	50.63	100	44.30	18.99	3.80	1.27	1.27	1.27	1.27	1.27

Table 9: Comparison of SIR Trend with Google Trend ($deviation = 0.4106$)

Although we cannot exactly replicate the Google Trend data, we were able to achieve the general trend with a sufficiently small deviation value. Thus we can conclude that the spread of information over our Clustom network is comparable to that of a real world network.

7 Conclusion

In the recent years, with the rising popularity of social media platforms, the spread of information has become more and more rapid. Information is now able to be easily spread from a single person to across the globe. In this paper, we have evaluated several different network models (Erdős–Rényi, Watts-Strogatz, Barabási–Albert, Mediation-Driven Attachment and Clustom Random Network Models), each of them utilizing different parameters and having different characteristics in the effort to simulate the nature of the real world human network. The main objective of this study is to understand and simulate the spread of information through the Facebook network. From the results, we were able to conclude that the Clustom Random Network Model is the best candidate to effectively mimic the nature of the Facebook network. Meanwhile by looking at the custom network graph generated, we could also see that it resembles the Facebook network with multiple closely connected clusters in the middle that are connected to a few smaller clusters at the sides. In addition, putting this model to the test of a simulation of information spread via the SIR method, we were able to closely match a Google search trend (Trump-Kim) with the 2 trends plotted having similar peaks and shapes with minimum error value. Thus we can conclude Clustom Random Model is the best network among the other 4 networks in representing the real world Facebook network.

8 Appendix

8.1 General Functions

```
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
import random
from pyswarm import pso

'''Function that prints all required data of a graph G.
(Plots network graph and vertex degree distribution histogram, then prints |V|,|E|,
average vertex degree, CC, number of triangles, average path length and diameter.)'''
def print_network_data(G):
    v_degrees = np.array(list(dict(nx.degree(G)).values()))
    nx.draw(G, node_size = 2.5 * v_degrees, node_color = 'blue',
    alpha = 0.2, edge_color = 'green')
    plt.show()

    vertex_degrees = list(dict(nx.degree(G)).values())
    plt.hist(vertex_degrees, bins = np.linspace(0, 300, 80),
            facecolor='blue', alpha=0.75, rwidth = 0.9)
    plt.title("Vertex degree distribution")
    plt.grid(True)
    plt.show()

    n = nx.number_of_nodes(G)
    m = nx.number_of_edges(G)
    cc = nx.average_clustering(G)
    triangles = np.trace(np.matrix((nx.to_numpy_matrix(G)**3))/6)
    try:
        avg_path_length = nx.average_shortest_path_length(G)
        diameter = nx.diameter(G)
    except: #in case graph is not fully connected
        avg_path_length = 'error'
        diameter = 'error'
    print("|V| =", n)
    print("|E| =", m)
    print("Average degree is", 2 * m / n)
    print("CC =", cc)
    print('Number of triangles = ', triangles)
    print('Average path length = ', avg_path_length)
    print('Diameter = ', diameter)
    return None
```

8.2 Code For Facebook Network

```
'''Retrieving Facebook network data and plotting it'''
FbData = np.loadtxt("facebook_combined.txt.gz")
FbData = FbData.astype(int)
F = nx.Graph()
F.add_edges_from(FbData)
print_network_data(F)
```

8.3 Code For In-Built Random Models (ER,WS,BA)

```
'''Creating and plotting the Erdős-Rényi network'''
E = nx.erdos_renyi_graph(4039,0.01081)
print_network_data(E)

'''Creating and plotting the Watts-Strogatz Network'''
W = nx.watts_strogatz_graph(4039, 44, 0.05)
print_network_data(W)

'''Creating and plotting the Barabási-Albert network'''
B = nx.barabasi_albert_graph(4039, 22)
print_network_data(B)
```

8.4 Code For MDA Random Model

```
'''Function for creating a MDA network graph'''
def mda(n,m):
    A = np.ones((m+1,m+1)) - np.diag(np.ones(m+1))
    B = np.zeros((n,n))
    B[0:m+1,0:m+1] = A
    for i in range(m+1,n):
        mediator=np.random.choice(np.array(list(range(i))))
        neighbor = B[mediator,:]
        neighbor_index = np.where(neighbor >0)[0]
        vec_rand = np.random.choice(np.size(neighbor_index),m,replace = False)
        a= neighbor_index[vec_rand[range(m)]]
        B[a,i]=1
        B[i,a]=1
    M = nx.Graph(B)
    return M

'''Plotting the Mediation-Driven Attachment Network'''
M = mda(4039,22)
print_network_data(M)
```

8.5 Code For Clustom Random Model

```
'''Function that splits the number of nodes, n, into g clusters in the main function'''
def splitn(n,g):
    listofsubgraphs = [1]*(g+1)
    #extra cluster because 0th cluster is too small to be significant
    for _ in range(n-g):
        listofsubgraphs[int(np.cbrt(np.random.randint((g+1)**(3))))] += 1
        #using y=x**3 to form a biased distribution
    for i in range(len(listofsubgraphs)-2,-1,-2):
        listofsubgraphs += [listofsubgraphs.pop(i)]
        #sort to get largest subgraphs to be in the middle
    return listofsubgraphs

'''Function for creating WS graphs' adjacency matrices in the main function.'''
def wsmatrix(n, l, p):
    k = int(n*(l**(n//500+1))/2)*2
    B = (np.arange(n).reshape(n, 1) - np.arange(n).reshape(1, n)) % n
    A = 1 * (((B >= 1) & (B <= k/2)) | ((B >= n - k/2) & (B <= n-1)))
    for i in range(n):
```

```
        v_maybe_rewired = (i + np.arange(1, k //2 +1)) % n
        v_rewired = v_maybe_rewired[np.random.rand(k // 2) < p]
        v_maybe_rewired_to = np.where((A[i, :] == 0) & (np.arange(n) != i))[0]
        v_rewired_to = np.random.choice(v_maybe_rewired_to, len(v_rewired), replace = False)
        A[v_rewired, i] = 0
        A[i, v_rewired] = 0
        A[v_rewired_to, i] = 1
        A[i, v_rewired_to] = 1
    return A

'''Function for creating Clustom network graph'''
def customrandomgraph(n, g, l, p, q):
    if n < g or p > 1 or q > 1:
        return 'error'
    else: n = splitn(n,g)
    ws = [wsmatrix(i,l,p) for i in n]
    listofclusters = [[] for i in range(g+1)]
    for h in range((g+1)**2):
        i = h//(g+1)
        j = h%(g+1)
        if j == i:
            listofclusters[i] += [ws[i]]
        else:
            distance = min(j-i,g-j+i+1)
            listofclusters[i] += [1*(np.random.rand(n[i],n[j]) < (q/distance**2))]
    A = np.triu(np.concatenate([np.concatenate(m,1) for m in listofclusters]))
    custommatrix = np.mat(A + A.T)
    return nx.Graph(custommatrix)

'''Function for plotting the Clustom network
m is the desired number of edges, e is the error range for number of edges.
Prints progress as it runs.'''
def plotcrm(n, g, l, p, q, m, e):
    difference = e
    while difference >= e:
        print('Finding graph,')
        try:
            print('creating adjacency matrix,')
            C = customrandomgraph(n, g, l, p, q)
            print('checking connectedness,')
            d = nx.diameter(C)
        except:
            print('error.')
            continue
        else:
            difference = abs(nx.number_of_edges(C) - m)
            if difference >= e:
                print('number of edges not optimal.')
    print('graph found, plotting graph.')
    print_network_data(C)
    return None

'''Plotting the Clustom Network'''
plotcrm(4039, 7, 0.27, 0.09, 0.0001, 88234, 3000)
```

8.6 Code For SIR

```
'''Generating network for SIR model'''
G = plotcrm(400, 7, 0.27, 0.09, 0.001,0,99999)

'''SIR function'''
def SIR(A, init, p, q, d):
    #A: adjancency matrix
    #init: number of infected nodes at day 0
    #p: probabilty of infecting those who are Susceptible
    #q: probability of recovering from being Infected
    #d: number of days

    G = nx.Graph(A)
    nx.draw_networkx(G)
    plt.show()
    n = len(A)
    Number_of_susceptible = [n-init]
    Number_of_infected = [init]
    Number_of_recovered = [0]
    print('Day 0')

    #create a list to contain nodes that are infected
    #initialise with random nodes that are Infected
    IList = np.array(random.sample(range(n),init))
    #create a list to contain nodes that have a chance of being infected
    SList = np.setdiff1d(np.arange(n),IList) #contain nodes that has not been Infected
    RList = np.array([]).astype(int) #initialise with empty list

    print('Number of Infected',init,', Infected List:',IList, '\n')

    #as the days go by from day 0 to day n+1, the lists will change,
    so we use a for loop to iterate each day
    for i in range(1,d):
        print("Day",i)
        #search through the infected list to check if each node has recovered
        for node in IList:
            if np.random.rand() < q:
                RList = np.append(RList,node) #add the recovered nodes to the Recovered List
        IList = np.setdiff1d(IList,RList) #remove recovered nodes from the infected list

        #finding infected neighbours of susceptible nodes
        B = np.copy(A)
        B[:,SList.astype(int)] = 0
        B[:,RList.astype(int)] = 0
        B[IList.astype(int),:] = 0
        B[RList.astype(int),:] = 0
        #the probability of infection for each susceptible node
        is based on the number of infected neighbours
        #loop through the susceptible list and append nodes that
        have been infected to the infected list
        for node in SList:
            #the probability of infection for each susceptible
            node is based on the number of infected neighbours
            Probability_of_infection = 1-(1-p)**(np.sum(B[node]))
            if np.random.rand() < Probability_of_infection:
                #add the newly infected nodes to the Infected list
```

```
                IList = np.append(IList,node)
        #remove the infected nodes from the Susceptible list
        SList = np.setdiff1d(SList,IList)
        Number_of_susceptible.append(len(SList))
        Number_of_infected.append(len(IList))
        Number_of_recovered.append(len(RList))
        #print("Susceptible:", len(SList), ", Infected:", len(IList), ", Recovered List:", len(RList))
        #print()

    #plt.plot(np.arange(d), Number_of_susceptible, label = 'Susceptible')
    #plt.plot(np.arange(d), Number_of_infected, label = 'Infected')
    #plt.plot(np.arange(d), Number_of_recovered, label = 'Recovered')
    #plt.legend(loc = 'center right')
    #plt.xlabel('Days')
    #plt.ylabel('Individuals')
    #plt.show()
    return Number_of_infected

'''Running SIR'''
RunSIR = SIR(A, 8, 0.6, 0.6, 12)
```

8.7 Code For PSO

```
'''Retrieving Google Trend data'''
Googletrend_KimTrumpData = np.loadtxt("TrumpKim12weeks.txt")
days = list(range(12))
google_search = Googletrend_KimTrumpData
A = (nx.adjacency_matrix(G).todense())

'''Modified SIR function to reduce computation time in PSO function'''
def PSO_SIR(A, init, p, q, w):
    G = nx.Graph(A)
    n = len(A)
    infected = [init]

    IList = np.arange(0,n,int(n/init)) #systematic sampling to partially seed results
    SList = np.setdiff1d(np.arange(n),IList)
    RList = np.array([]).astype(int)

    for i in range(1,w):
        for node in IList:
            if np.random.rand() < q:
                RList = np.append(RList,node)
        IList = np.setdiff1d(IList,RList)

        B = np.copy(A)
        B[:,SList.astype(int)] = 0
        B[:,RList.astype(int)] = 0
        B[IList.astype(int),:] = 0
        B[RList.astype(int),:] = 0

        for node in SList:
            Probability_of_infection = 1-(1-p)**(np.sum(B[node]))
            if np.random.rand() < Probability_of_infection:
                IList = np.append(IList,node)
        SList = np.setdiff1d(SList,IList)
```

```
        infected.append(len(IList))

    maxvalue = max(infected)
    normalised_infected = np.array(infected)*(100/maxvalue)
    trend = Googletrend_KimTrumpData
    difference = np.sqrt(sum(((normalised_infected-trend)/100)**2))
    #each value is taken as a percentage to keep difference small
    print('p = %f, q = %f, difference = %f'%(p, q, difference))
    print(normalised_infected)
    return difference

'''Function used in the PSO function'''
def f(x): #x = [p,q]
    p = x[0]
    q = x[1]
    difference = PSO_SIR(A, 8, p , q, 12)
    return difference

'''Running PSO'''
lb = [0.2, 0.5]
ub = [0.4, 0.7]
pq_value, distance = pso(f, lb, ub)
p = pq_value[0]
q = pq_value[1]

'''Code for printing the Google Trend and optimized SIR trend'''
distance = 1
while distance >= 0.6:
    plt.title('Google Trend v.s. SIR Trend')
    plt.plot(days, google_search, color='g', label = 'Google Trend')
    plt.legend(loc = 'center right')
    distance = PSO_SIR(A, 8, p, q, 12)
```

References

[1] Stanford Large Network Dataset Collection, https://snap.stanford.edu/data/ego-Facebook.html

[2] Stanley Milgram, "The small world problem", Psychology Today, 1967.

[3] Md. Kamrul Hassan, Liana Islam, and Syed Arefinul Haque, "Degree Distribution, Rank-size Distribution, and Leadership Persistence in Mediation-Driven Attachment Networks", 2016.

[4] https://trends.google.com/trends/explore?geo=SG&q=Trump-Kim

[5] Kennedy J., Eberhart R, "Particle swarm optimization", Proc. IEEE Int. Conf. Neural Network, 1995