Emergence of Segregation and Affiliation Switching in Mixed Populations and Social Networks

Conor Cullen, Duncan Thorley, Libby Walker, Maximilian Fullerton

May 4, 2018

Abstract

Society can be viewed mathematically as a network of people connected in different ways. The way in which these networks interact with different stimuli can be exploited if understood. A particular enterprise media companies are interested in is polarisation of audiences, in order to maximise their following. The objective is to explore methods of how to achieve a separation of followers based on their preferences and the people they share connections with. Here we show how segregation emerges - if at all - in different types of social networks, and investigate the effect of changes in affiliation switching and connectivity on the network, as well as the convergence of the "social process" as defined by a Markov chain, which has been adapted from considerable literature on the subject. This project illustrates how segregation becomes apparent and interestingly how long term behaviour as well as short term indicates affiliation switching in many conditions. We see that a scale-free type network takes longer for segregation to emerge than other types of network, and how the frequency of short attribute distances indicate segregation between individuals.

1 Introduction

The media's main objective is to capture and retain an audience. They implement this by focusing their attention on social networks in society, polarising views. Some commonly targeted views are politics, sport, and culture. The idea is to have a unique viewpoint from all rival media companies and so by indoctrinating the population with their views they will become the solely sought after news outlet.

The population starts off as being well connected, having access to a variety of views. The media are known as hubs as they have many connections compared to individuals in the underlying network. As they impose their views, they take grasp of the population and separate it. A common type of segregation is the strong divide between the left and right of politics, which is known as polarisation: a special case of segregation in which there are only two options involved. In this project, we will look at polarisation as an example of segregation. The models will generate two large degree nodes in a network of individuals, known as hubs, representing two opposite ideas (e.g. left and right). A connection between an individual and a hub implies that the individual is aligned with the ideas of that particular hub.

Thomas Schelling published "Dynamic Models of Segregation" in 1971 [1], proposing a simple dynamic model of racial segregation. The Schelling Model models the world as a grid, with each cell representing a house. They are occupied by two types of "agents", labelled red and blue, in equal numbers. 10% of cells are empty. An agent may be satisfied or dissatisfied at any point in time depending on the type of their neighbouring agents. The simulation turns unsatisfied agents into satisfied ones by moving their neighbours and stops when all agents in the grid are satisfied and the grid is thus segregated into two types. This phenomenon is actually reached quite quickly. Clusters of similar agents appear and coalesce over time. Schelling's model could provide evidence for racism due to the

segregation of different types of people, however it provides a strong argument about the relationship between a system and its types. Schelling's Model gives an example of a possible cause of segregation, but no explicit causes of the phenomenon, which this report tries to justify.

Representing the population alternatively through a Markov Chain based on the probabilistic social process model allows for association between the population to be modelled and investigated. Considering the time in which it takes a network to segregate and the type of segregation that prevails is important to explore. Different outcomes are indicative of different situations.

2 Methods

2.1 Model 1: Simple Network Hub Model

One possibility for a segregation model can be produced if one assumes that both network hubs are exactly the same except for the position they take. If both hubs are connected to all individuals in a population and both generate the same amount of information, then it can be assumed that there is an equal chance for an individual to remain with its current alignment or to switch.

Though this model may appear quite basic, complexity is introduced when using environmental factors to affect the decision making process. For example, if an individual were surrounded by people of like-minded alignments then their position would likely be reinforced, making it harder for the individual to switch position. Conversely, if that individual were surrounded by people of a different alignment then the pressure on them to change alignment would be much greater. This seems representative of real life and, although data is difficult to obtain, makes for an assumption that is within reason.

Connections between individuals might change throughout this process. No individual is connected to every other individual so it is possible that connections might be established or broken with each passing iteration. Though this might be random, the chance of this happening could be affected by the alignments of individuals on either end of a connection (if two people have different opinions then the connection could be more likely to change). In the real world, changing connections occur all the time. Networks of people are constantly changing with new connections being established. Though these might not be random, assuming that changes in connections is random is another fair assumption to make.

Though the network hubs might be connected to every individual in this model, this isn't necessarily the case for the individuals. Given a network of 100 individuals, for example, the average number of connections to other individuals might be quite low at 20 percent. If an individual has too many connections then connections will be more likely to break even if both individuals are of the same alignment. Likewise, if an individual has no connections then they would be more likely to start new connections.

As mentioned previously it is possible that the alignments of individuals will affect the probability of changes occurring. If we assume that this occurs throughout the model then this should represent the real world where people make biased decisions that are less random.

2.1.1 Assumptions Made

- Network hubs are identical
- Alignment peer-pressure
- Changing connections

- Limit to connectivity
- Bias on decision making

2.1.2 Process

To determine if a node will change alignment we decided to implement a Beta distribution to generate a random number using the connections to that node. To prevent alignments from switching too quickly, a margin in the centre of the distribution is created. Values drawn from this region will not cause the node to switch alignment.

Using a Beta distribution between 0 and 1 and with "a" and "b" as variables, where "a" is the frequency of one alignment and "b" is the frequency of the other, we can randomly generate numbers with a varying likelihood depending on the surrounding alignments, with the following p.d.f:

$$p(x) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)} \qquad (1)$$
$$where$$
$$B(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$$

For a given number of elements, a network is produced and the "Connectivity" is input. The connectivity matrix and alignment list is pseudo-randomly generated before the program iteratively updates the connections and weights for a given number of steps. The initial and final matrices plus the lists are printed for comparison.

Having produced a model with inputs to account for different situations, it is now possible to test a variety of scenarios. The model is capable of running smoothly for up to 40 individuals so the tests will consist of scenarios with less than 40 nodes, with a variety of different connectivities and conversion rates.

2.2 Model 2: Social Network Model

2.2.1 Assumptions Made

- The most simple model of a network is fully connected, assuming that all individuals are connected.
- An edge connected between an individual and a hub means they are affiliated with that hub
- Once a node loses its neutrality (i.e. becomes aligned with a hub), they cannot go back to being neutral

The two social networks we used in the development of our models are the Erdos-Renyi (random) network [2] and the scale-free network as well as a baseline fully-connected network.

The bipartite graph representing each network is defined by $B(V, C, M)$, where V is the set of nodes of the underlying network G, C is the set of hubs, and M is the set of edges.

2.2.2 Social Network Types

2.2.2.1 Fully Connected Network

The most basic model is one in which every individual in the population is connected to all other individuals. An example is shown below that includes the two different hubs, represented by red

and blue nodes, and individuals in the population, shown in green, before beginning the segregation simulation:

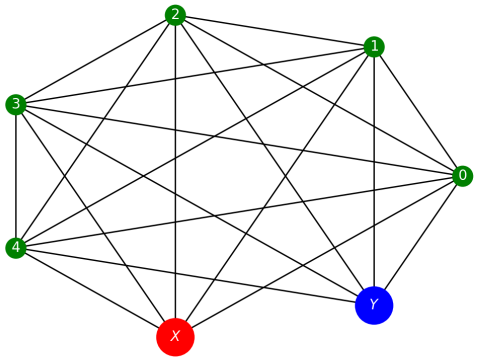


Figure 1: Example of a fully connected network of a population of size 5. Green nodes correspond to neutral individuals from the set V and the nodes labelled X and Y represent the hubs from the set C

2.2.2.2 Erdos-Renyi Network

Our random (Erdos-Renyi) network was defined as G(N,p), such that each pair of N labelled nodes is connected with probability p. The degree distribution of a random network follows the Poisson distribution, e.g. the probability that a randomly chosen node has degree k is given by the following:

$$p_k = e^{-<k>} \frac{< k >^k}{k!} [3] \tag{2}$$

here $< k >$ is the average degree of a random network and is given by $< k >= p(N-1)$. This serves as the "scale" of the network as most nodes have comparable degrees within a small range of $< k >$.

The figure below shows an example of a random network of individuals along with hubs labelled X and Y:

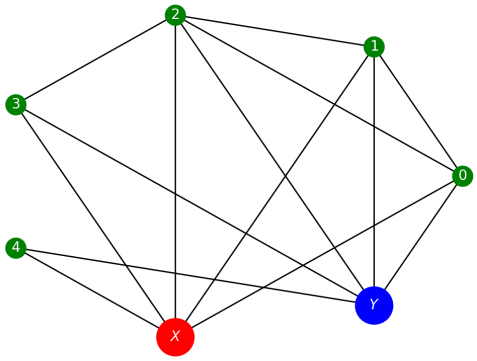


Figure 2: Example of an Erdos-Renyi network with a population of size 5, where the probability of a node forming a connection with an individual is 0.2 (note that the nodes have a minimum degree of 2 as they are initially connected to both hubs). Green nodes correspond to neutral individuals from the set V and the nodes labelled X and Y represent the hubs from the set C

2.2.2.3 Scale-Free Network

The Scale-free network has a Power-law degree distribution, so the probability that a node has degree k is given by:

$$p_k \approx k^{-\gamma}[3] \tag{3}$$

where γ is the Scale-free network degree exponent, or the probability for adding a new node connected to an existing node chosen randomly according to the out-degree distribution.

The figure below shows an example of a scale-free network of individuals along with hubs labelled X and Y:

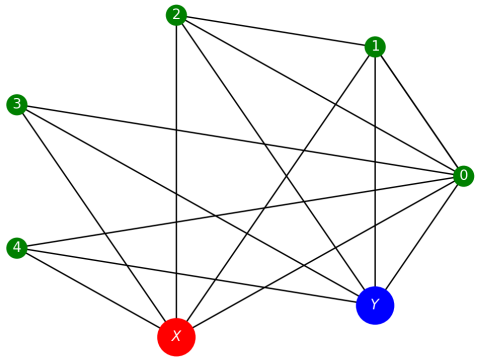


Figure 3: Example of a scale free network with a population of size 5, with $\gamma = 0.05$ (note that the nodes have a minimum degree of 2 as they are initially connected to both hubs). Green nodes correspond to neutral individuals from the set V and the nodes labelled X and Y represent the hubs from the set C

2.2.3 Transition States

The nodes in the network are each given initial types to distinguish their alignment. A green node in the set V has type 0, where it is connected to both hubs and has neutral opinions. A red node in the set V has type 1, where it is connected to the hub labelled X (where $X \in C$), which is also coloured red, indicating that the individual aligns with hub X's views. A blue node in the set V has type 2, where it is connected to the hub labelled Y (where $Y \in C$), which is also coloured blue, indicating that the individual aligns with hub Y's views.

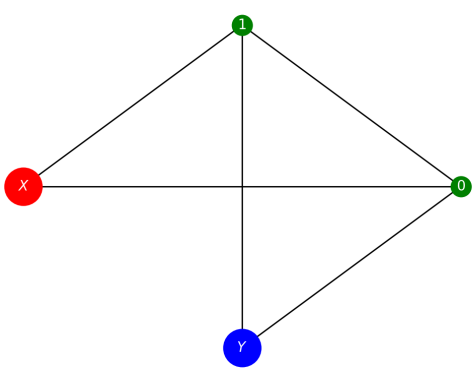
2.2.3.1 Join Transition

The join transition corresponds to node u joining hub $c \in C$, where u is a type 0 node. We choose a node-hub edge $(u, c) \in M$, add it to M' (i.e. $M' = M \cup (u, c)$), and remove the other hub edge connected to u. Effectively, this is the decision to initially choose one opinion over another. Let $B(C, V, M)$ be the bipartite graph with the definitions as before, and an edge $(u, c) \in M$ means that node $u \in V$ is connected to hub $c \in C$. Let also $\{p_c\}$ be a set of probabilities for all $c \in C$. Given $B(C, V, M)$ and $\{p_c\}$, the model generates a graph $G(V, E)$ by creating edge (u, c) with probability $p(u, c)$:

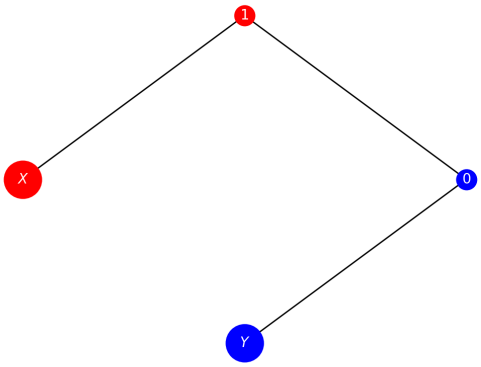
$$p((u, c)|u = type0) = 1 - \prod_{k \in C}(1 - p_k)[4] \tag{4}$$

where $k \in C$ indicates that k is a member of C (i.e. one of the two hubs). A node u that is not connected to any $v \in V$ has zero probability of joining a hub.

Figure 4: Join Transition



(a) Before join transition



(b) After join transition: node 1 has joined hub X and node 2 has joined hub Y

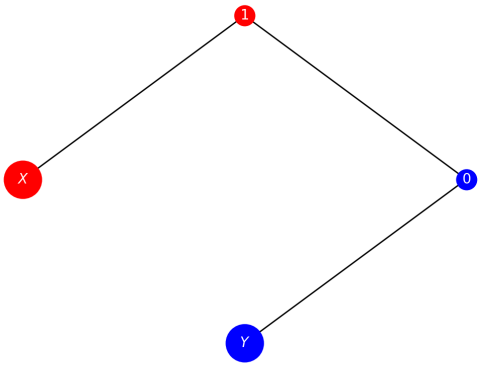
2.2.3.2 Switch Transition

The switch transition corresponds to node u switching from hub $c \in C$ to hub $c' \in C$, where u is a type 1 or type 2 node. We choose node-hub edge $(u, c') \notin M$, add it to M' (i.e. $M' = M \cup (u, c')$), and remove the edge $(u, c) \in M$. This is the decision that an individual makes to switch alignment. The probability of an individual choosing to switch from one hub to another is:

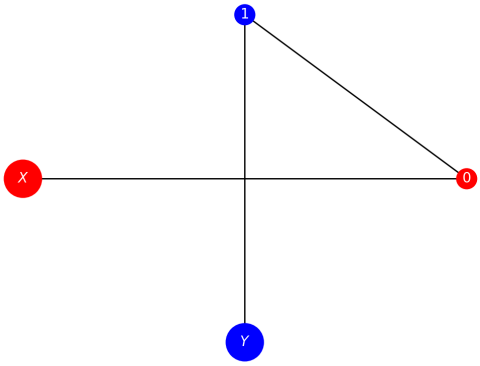
$$P((u, c')|u = type1 \cup u = type2) = 1 - \prod_{k \in C} (1 - p_k) \tag{5}$$

where $k \in C$ indicates that k is a member of C. Again, a node u that is not connected to any $v \in V$ has zero probability of joining a hub.

Figure 5: Switch Transition



(a) Before switch transition



(b) After switch transition: node 1 has switched to hub Y and node 2 has switched to hub X

2.2.4 Weight Update Methods

Initially, the nodes are given weights 1 or -1 distributed randomly (fifty-fifty chance), and if an edge is chosen connected to a node that is type 0 then the weight will determine its new type. If the node

has weight 1, it will become type 1, and if it has weight -1, it will become type 2. The weights are updated after each time step, again being distributed with probability 0.5.

We alter the method of updating these weights to investigate the effect on segregation. Firstly, one way is to increase the probability of choosing weight 1 and decrease the probability of choosing weight 2 by the same amount at each time step, when eventually the probability of choosing weight 1 will be 1. We expect that this will slowly increase the size of the type 1 group and decrease the size of the type 2 group, resulting in a biased segregation. Secondly, another method will be for each individual to assess their neighbours (i.e. other nodes in the network an individual is connected to) and make a judgement to update to weight 1 if most neighbours are type 1 and update to weight -1 if most neighbours are type 2. We expect that this will show the effect that connectivity and conformity bias have on the segregation process.

2.3 Model 3: General Model of Network Segregation

We begin this model by outlining a general Markov-chain model of network evolution that operationalises the aversion aspect of the Schelling segregation model [1], which focuses on the links between individuals based on their alignment. Major components to the model include the initial network conditions, including the assignment of attributes to network individuals, as well as the basic process of aversion-driven network rewiring [5].

Each vertex is assigned a fixed attribute that will be used to assess their similarity to other network individuals. Vertex attributes are represented by the function $\omega : V \to [-1, 1]^r$, where $r \in \mathbb{N}$ represents the dimensionality of attributes. The attribute distance between any two vertices in the network is defined as $d : [-1, 1]^r * [-1, 1]^r \to [0, 1]$. Attribute distances are continuous as they are uniformly randomly distributed with the following probability density function:

$$p(d) = \frac{1}{b - a} \tag{6}$$

anywhere in the interval [a,b), and 0 elsewhere. This is done by giving each edge in the network a "weight" equal to d.

Large attribute distances are "long edges" (d$>$0.5) and small attribute distances are "short edges" (d$\leq$ 0.5). An individual's inclination to cut ties with a dissimilar individual is represented by a single parameter called aversion bias, represented by $p \in (0, 1]$. Larger values of p indicate that individuals have a strong aversion (i.e. disinclination) to maintain links with individuals who are very different from themselves.

2.3.1 The Network Evolution Process

The social process analysed here $[\mathfrak{B}(G, p, r, \omega, d) = (G_t)_{t=0}^{\infty}]$ is defined as a Markov chain and generates a stochastic sequence of graphs. The process begins at $t = 0$, with $G_0 = G$. Time step t, for $t \geq 1$, is defined to be the transition between G_{t-1} and G_t.

The stochastic process of evolution occurs through link termination, followed by random "rewiring" (i.e. similar to the transition states defined above, except between individuals). At each time step, a random edge is chosen between individuals u and v. The individuals assess their attribute distance $d(\omega(u), \omega(v))$ and make the choice to keep or delete their links based on the attribute distance and p. The probability of tie deletion between u and v is $d(\omega(u), \omega(v)) * p$. When a tie is deleted, it is then rewired by removing uv, choosing a vertex $x \in V$ uniformly at random, then finally adding ux or vx with equal probability.

2.3.2 Measuring Network Segregation

Attribute distance can be thought of as a discrete variable by fixing $K \in \mathbb{N}$ and partitioning all distances into K discrete bins. Let $E_i(t)$ (where $i \in \{1, 2, ..., K\}$) denote the total number of edges in G that have attribute distance $\frac{i}{K}$. Then:

$$E_K(t) = |E(G)| - \sum_{i=1}^{K-1} E_i(t)[5] \tag{7}$$

Next, let $F(t)$ represent the total length of all edges in G: the probability of rewiring an edge will depend on this quantity. We have:

$$F(t) = \sum_{uv \in E(G)}^{K-1} d_K(\omega(v), \omega(u))[5] \tag{8}$$

We determine the limit of the process when a stationary distribution of the Markov chain is obtained.

The system of ODEs in the general case (for any $K \in \mathbb{N}$) can be written as:

$$\frac{dE(x)}{dx} = \frac{-p}{K^2}(AE(x) - B)[5] \tag{9}$$

where $A = (a_{ij})$ is a matrix defined as follows: $a_{ij} = K - j$ if $i \neq j$ and $a_{ii} = (K - i) + iK$ otherwise, $E(x) = (e_1(x), e_2(x), ..., e_{K-1}(x))^T$ and $B = (K, K, ..., K)^T$ are vectors.

One can solve the characteristic equation to find the eigenvalues $\lambda_1, \lambda_2, ..., \lambda_{K-1}$ of the matrix A. Then the general solution is of the form:

$$e_i(x) = \frac{1}{i \sum_{j=1}^{K} \frac{1}{j}} + \sum_{j=1}^{K-1} C_{i,j} exp\Big(\frac{-p}{K^2}\lambda_j x\Big), [5] \tag{10}$$

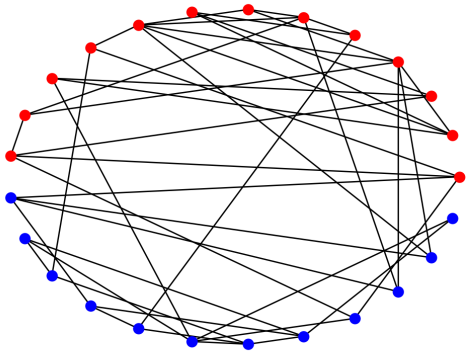
for $i \in \{1, 2, ..., K-1\}$, and where constants $C_{i,j}$ depend on the initial values to the system of ODEs. More detail can be found in [5], but an attempt at replicating the model is included in the results below.

3 Results

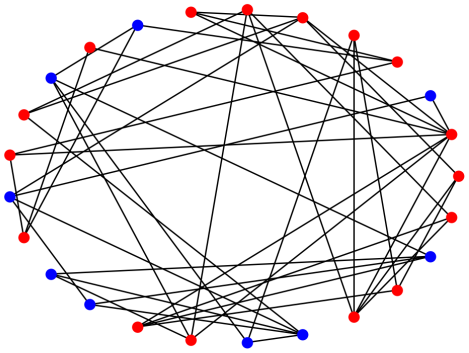
3.1 Model 1: Simple Network Hub Model

For segregation to form in a network, there must be a limit to the number of connections each node can have. The more connection each node has the more likely that one node will connect the others to other groups. As such we will run the model starting with 25 nodes and low connectivity with small and large conversion margins before raising the connectivity. 25 nodes is sufficiently large to ensure that segregation can form between two different alignments, as well as within each alignment. By using low and high conversion margins we can also watch the network evolve differently over time, providing insight into how a network hub might want to polarise individuals based on their opinion. Each scenario is run multiple times and the trends are tracked.

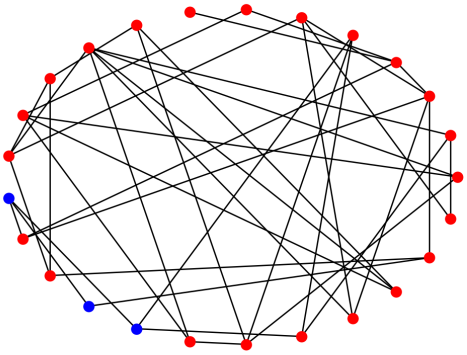
Figure 6: Starting at 5% connectivity, the network is initially entirely segregated with blue and red nodes having no common links with the exception for a couple of nodes. With a margin of 0.45, it's much more likely for nodes to convert even with no ties to the opposite alignment. As such some very slight inter-alignment connections occur, though these connections either result in a conversion of one node or break. After approximately 40 steps one alignment typically begins to dominate the network before fully converting in approximately 60 steps.



(a) Starting point (t=0)



(b) Mid point(t=25)



(c) End point (t=50)

With a margin of 0.7 it's significantly less likely for nodes to convert and this is represented in the model with the shift in network alignment occurring much later and segregation occurring within approximately 90 steps.

For a connectivity of 10% the model follows the same trend as the 5% connectivity. The network starts similarly, but with proportionally more connections in fig. 6a. For the small conversion margin (0.45) one weight of node starts to take over the network at approximately step 40 in fig. 6b before fulling converting the network in 60 steps as represented in fig. 6c.

Figure 6 shows the network under 10% connectivity with a margin of 0.7. There are connections between the two weightings but the network is still significantly segregated to start with. The network begins to fully convert to one weighting at approximately step 30 and finally converts entirely by step

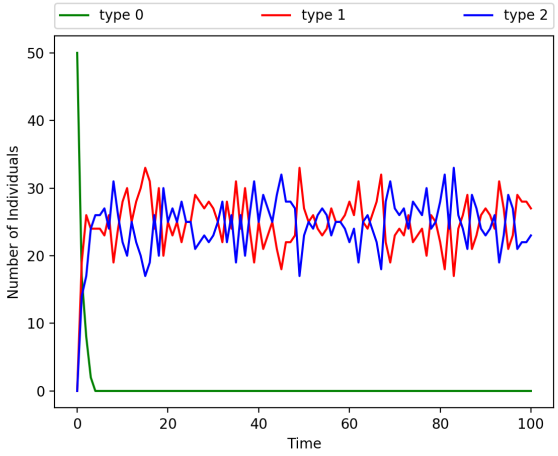
60. In fig. 6b it is clear that there is a greater number of red nodes than blue, though under examination the blue nodes are typically still only connected to other blue nodes. These means segregation is maintained even whilst the network begins to convert.

The exact same trend is then replicated with 15% connectivity; small conversion margins of 0.45 lead to a full conversion in 30 steps and a large conversion margins of 0.7 lead to a full conversion in 50 steps.

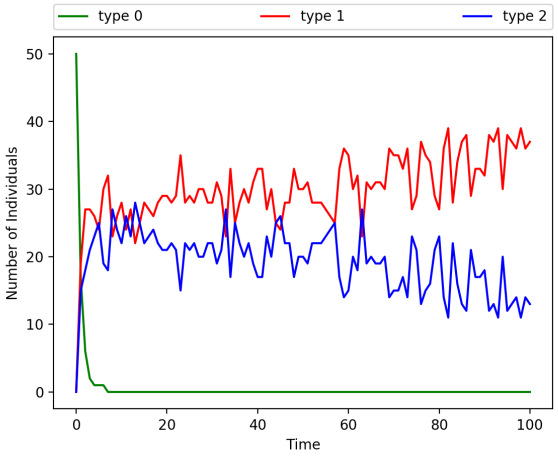
For 20% connectivity and a large margin (0.7) the model doesn't convert fully to one weighting or another. Although this is different to the other scenarios, the behaviour of the network remains similar. Proportionally, each node is mostly connected to nodes of the same weighting which indicates a level of segregation exists within the model. A small conversion margin of 0.45 does fully convert in 30 steps, with the network beginning to shift towards one weighting within 15 steps.

3.2 Model 2: Segregation in Social Networks

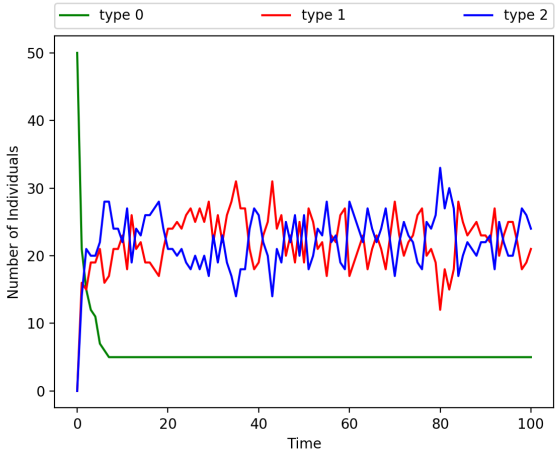
Figure 7: Segregation in different networks



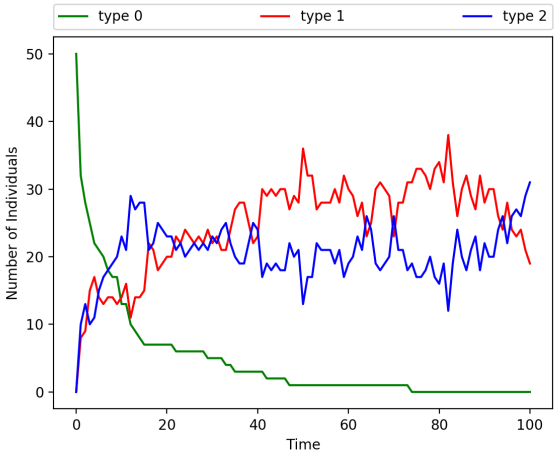
(a) Fully Connected Network with random weights



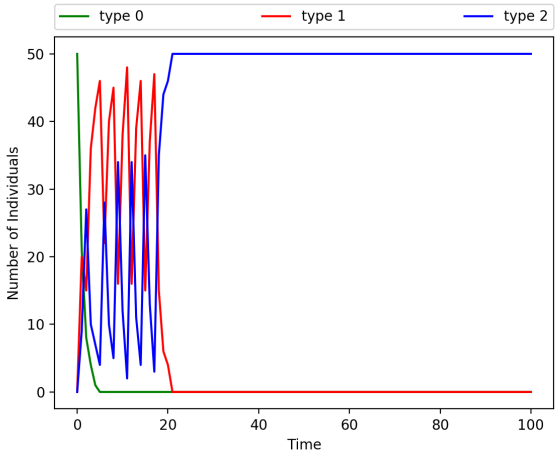
(b) Biased Fully Connected Network where P(type1) increases by the same amount P(type2) decreases in each time step

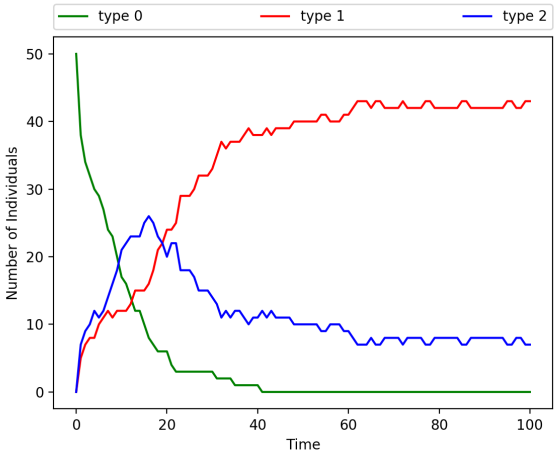


(c) Erdos-Renyi Network with p=0.05 and random weights



(d) Scale Free Network with random weights





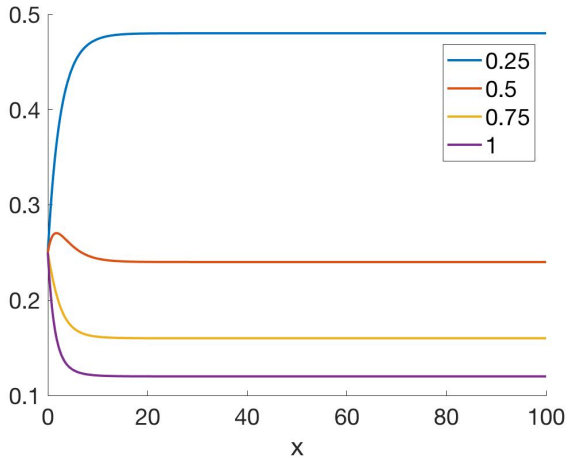
(e) Fully Connected Network where individuals align with the hub that is most common to their neighbours

(f) Scale Free Network where individuals align with the hub that is most common to their neighbours

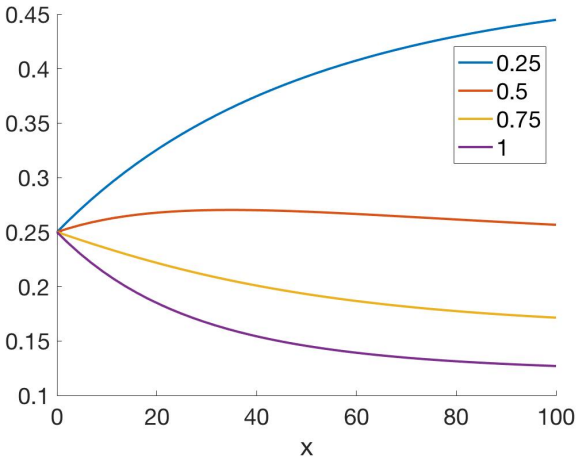
The graphs shown are produced over 100 time steps given the transitions from before with different networks for a population of 50 individuals. The graphs also demonstrate the use of different weight update methods, also shown in the methods section.

3.3 Model 3: Convergence of Markov Chain

Figure 8: Markov convergence scenarios with K=4. Horizontal axis represents steps in the Markov process and the vertical axis represents the proportion of edges of a given length. Labels on curves represent the length of the respective edges

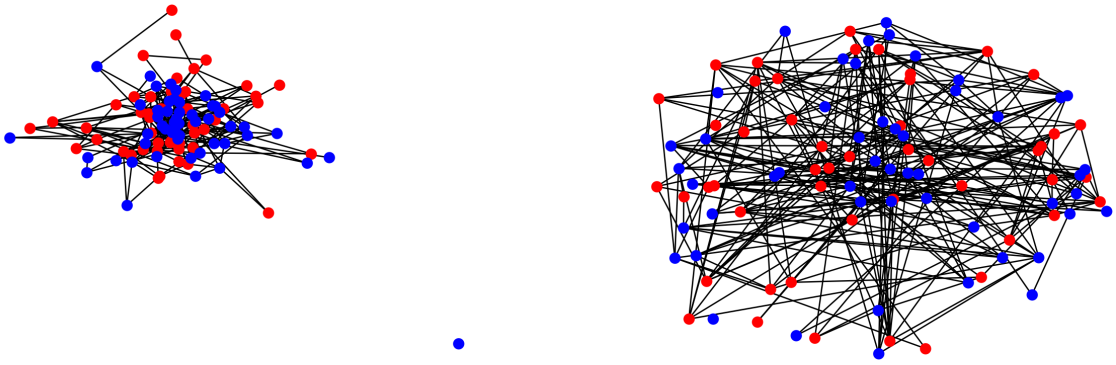


(a) Random graph (Aversion bias=1)



(b) Random graph (Aversion bias=0.05)

Figure 9: The emergence of network segregation where the aversion bias is 1. Left shows the initial clustered network and right shows the network as it has reached the stationary distribution of the Markov chain



(a) Random network (probability of a pair of nodes forming an edge=0.05) containing 100 individuals

(b) Network after 10000 iterations

4 Discussion

4.1 Model 1: Simple Network Hub Model

What the results seem to reflect is that when a network has fewer connections between individuals segregation is much easier to form. This also reflects our initial ideas about how segregation forms in a network. The more connections a network has the more likely it will allow for connections to form between both sides. When the margin within the beta distribution is increased from 0.45 to 0.7 the network achieves a state of full conversion slower. This is because changes within the network happen slower.

What this might mean in the real world is that a network hub looking to segregate a network of individuals wants to divide the network into smaller groups and to try to polarise the alignments. This makes it harder for individuals to create new connections to other individuals, limiting the spread of new opinions to other individuals and making it harder for individuals to change position. Conversely if a network hub wants to break up the segregation in a network then it should try to increase the number of connections an individual has with others and by trying to produce information that is less polarising.

4.2 Model 2: Segregation in Social Networks

Figure 7 illustrates how the different network types evolve. fig. 7a and fig. 7b demonstrate how being fully connected enables individuals to choose affiliations quickly, shown by the rapid fall of the green line. Conversely, fig. 7c and fig. 7d indicate a longer time until all allegiances are formed; in particular fig. 7d takes a very long time before allegiances are made. The reason for this is because in a scale free network, some individuals have more connections than others, so the probability of an edge being chosen is higher for one of these high degree nodes, meaning it will take longer for individuals with less connections to choose an affiliation.

Figure 7b represents a bias towards individuals choosing hub X over hub Y. As expected, when the weight towards type 1 becomes strengthened over each time step, and the weight towards type 2

decreases by the same amount, the type 1 group of individuals slowly becomes more dominant.

Figure 7c shows that a random network is similar in nature to a fully connected one. In this figure, the probability of each node forming an edge with a neighbour is 0.05, so the network will have about 5 individuals that will not align with either hub X or Y. This is indicated by the green levelling out after about 10 iterations, after which the only transition taking place is random switching of affiliation of type 1 and type 2 nodes.

Figure 7e and fig. 7f show the effect of controlled affiliation switching as opposed to random switching. The probability that a node switches affiliation depends on the affiliation of its neighbours. In fig. 7e, it is clear that this is the only graph that reaches a convergence within the 100 time steps. The reason for this is that over time, the probability will lean towards one hub (in this case hub Y) more than the other because at one point (around t=20) it is apparent that any nodes that are type 1 have more neighbours that are type 2 than neighbours that are also type 1. Hence, these individuals will switch to hub Y until all are affiliated with hub Y. However, fig. 7f depicts a more gradual process, which is again due to the inter-network hubs (i.e. large degree nodes in G), which makes affiliation switching less erratic due to some of the nodes having few connections, meaning they have a lower probability of aligning with a hub based on their neighbours' affiliations.

4.3 Model 3: Convergence of Markov Chain

The results depicted in fig. 8 show the solution to the ODEs introduced by eq. (10), in a random network with strong aversion (p=1) and weak aversion (p=0.05). The network that has a weaker aversion bias clearly takes longer to reach the stationary point than the fully connected network. Figure 9 shows how the network becomes more segregated after many time steps, as the edges are deleted and rewired based on attribute distance (i.e. an edge with a greater distance $d(\omega(u), \omega(v))$ has a greater probability of being rewired). This relates to fig. 8 because the proportion of shorter edges increases (in this case 0.25) as the proportion of other edge length decreases. Attribute distance measures the "closeness" of individuals, so this means that individuals will delete ties with those whom they don't share a strong connection and keep links with those they do share with. The network with the larger aversion bias takes a shorter time to reach the stationary point because individuals have a strong aversion to maintaining links with individuals who are dissimilar to themselves (i.e. short edges), whereas with a smaller aversion bias individuals have a smaller probability of removing connections so the curves take longer to reach a stationary level.

5 Conclusion

This project has given a reasonable illustration of network segregation and affiliation switching. Through the use of different models, different aspects of segregation have been explored. Looking at the different models and considering average time until segregation is reached, it is evident that the more connections there are, the faster the rate of segregation is. However, Model 1 conversely illustrates that for segregation to be maintained less connections are favourable. Affiliation switching is reflected both in the models and in real life which provides provenance to our models.

The three alternate approaches to this problem explore different areas of segregation. Model 1 demonstrates how best to perpetuate segregation, Model 2 shows how different types of networks form segregation, and Model 3 examines how individuals in a network react to others' views. Each model can be exploited for simulating world events.

Applying this research to the world would be useful for modelling political views and media audiences. Implementing the code created for a real life situation would require a lot of simplification from

the real world, however it would still be useful to see how different initial conditions affect the populations response. A large media conglomerate could use this research in order to tailor their material and political standing to maximise profits.

5.1 Merits and Limitations

The models in this report are very much over-simplifications of any real situation hence they are somewhat limited. Moreover, comparison between methods is difficult as they explore different approaches to segregation. However, they each illustrate segregation whilst demonstrating different features of the phenomenon such as affiliation. Furthermore, the merit of having a basic model is that it is easy to build on and apply to many situations, and provides a better understanding of the processes that give rise to network segregation.

The Simple Network Hub Model considers the progression of a network that starts in segregation, the factors that will improve the segregation of the network and the behaviour of nodes/individuals. The model doesn't describe how to take a population of random weightings and connections and segregate them. The model also uses discrete measurements to describe alignments, which is not representative of the real world. The model could be improved to incorporate a continuous distribution that works with the beta distribution, but in the time frame given this was not feasible. Finally, the assumption that the Network Hubs output the same amount of information could have been removed by altering the Beta Distribution function.

The second model explores affiliation switching over long time. This is interesting as it demonstrates segregation in a way likely to mirror real life, however this model doesn't allow for polarisation. Figure 7e shows how to dominate a population completely but not polarise. Moreover, this model doesn't consider the behaviour that once people form an opinion they can then become unaffiliated, so the green line would not tend to zero in reality. Nonetheless, this model used multiple population distributions and as such is versatile and the results it yields are useful for predicting population behaviour.

A large limitation is that the results in fig. 9 are somewhat irrelevant and the process does not reach the stationary point we were hoping to find. The results should show how the red and blue nodes are segregated into two separate groups. The aversion bias of 1 means that this should happen fairly quickly, however after 10000 steps there was not much change except for the edges becoming longer. The desired effect was for there to be more short edges than long edges, as shown by the solution to eq. (10) in fig. 8. Due to the aversion bias, individuals are more likely to be connected by a short edge to someone of the same type. The mistake here could have been an error when trying to replicate the model from [5], as the process that generates fig. 9 distributed the nodes randomly rather than according to edge distance. In layman's terms, this means the red nodes should be individuals with distances less than the population mean and blue nodes individuals with distances greater than the population mean. Another implication of this model is that attributes being distributed uniformly is unrealistic. If these attributes are represented by race, social status etc. then it is unlikely that this model is a viable representation.

5.2 Model Improvements and Further Steps

Some improvements to advance our research could be considered, such as including more population distributions to represent society more closely or introducing a third hub to measure any change to the segregation process. Also, creating some real world scenarios like having major events within the model and having a more complex hierarchy of influencers would make the model more useful for simulating the world. Moreover, changing the initial conditions from those given in the task to include a less well connected network to start could also be an interesting context to explore and see if there are any settings in which segregation is very difficult. The third model could also use some improvement to

reach a valid conclusion about network segregation according to aversion bias.

6 Further Insights

It is intuitive that someone will engage with a service that they feel reflects and matches their own stances. As a result companies have adapted so that they cater to certain sections of society. In monopolising their influence within a particular circle and pandering to the desires of their audience they can build a loyal set of individuals, consequently creating a secure stream of revenue. This is a process that consists of analysing data to predict the behaviour, interests, and opinions held by specific groups of people and then serving them the messages they're most likely to respond to. It is a technique that partially accredited to Donald Trump's success in the last American election. A sizeable portion of voters were identified as prioritising foreign affairs and national security that felt they were not being listened to. This underlying feeling held by many voters was taken advantage of with carefully placed pieces of propaganda in order to grow and nurture its importance in the presidential campaign and cause segregation between the two sides. In actual fact the fight was nowhere near as binary and divisive as the media made out but by only exposing people to the two extremes of the argument they forced society to polarise itself. This report seeks to understand the dynamics of how networks interact with each other to show what effects the media can have on segregating society.

One particular model we can analyse is the American political system and how it might be modelled in more detail. In reality this example will again consist of many different models due to the make-up of the nation. Network connectivity will depend on factors like geographical location and level of interaction with the media or on-line presence. Aversion bias and conversion bias will have a range of values throughout the population depending on their initial political stance. A large section of supporters on either side of the debate will be completely unaffected by any media or network influences as their political stance will be heavily entrenched on one side.

As our model suggests the fewer connections between individuals are the less connected a network is and the easier it is to segregate. This is why social media platforms that only expose (or at least more heavily weight) individuals to views similar to their own resemble less connected networks that lead to segregation. It is also why controversy has followed the recent role of social media in the political sphere. The digital era has welcomed in new techniques and an ability to take advantage of an audience unrivalled by anything society has seen before. Information now allows companies to understand exactly how people make their decisions and what is most likely to influence them. The power to control society to a greater degree than ever highlights the ease at which companies might exploit society and the responsibility they now wield. It is in society's interests to further our understanding and research in this field to gauge a company's motives when using this power and hold them accountable to their actions.

References

[1] F. McCown, "Schelling's model of segregation." http://nifty.stanford.edu/2014/mccown-schelling-model-segregation/, (Accessed: 5/4/18).

[2] J. Hopcroft, "Erdos-renyi model." www.cs.cornell.edu/courses/cs485/2006sp/lecture%20notes/lecture1.pdf, (Accessed: 5/4/18).

[3] A.-L. Barabási, "Network science." http://networksciencebook.com/, (Accessed: 5/4/18).

[4] J. Wang et al., "Community-affiliation graph model for overlapping network community detection." https://www-cs.stanford.edu/~jure/pubs/agmfit-icdm12.pdf, (Accessed: 5/4/18).

[5] A. Henry et al., "Emergence of segregation in evolving social networks." http://www.pnas.org/content/pnas/108/21/8605.full.pdf, (Accessed: 6/4/18).

Appendices

Simple Network Hub Model

```
from random import *
import numpy as np
import scipy.stats as spst
import matplotlib.pyplot as plt
import pylab as pl
import networkx as nx

N = int(25)        #Number of Nodes
C = float(10)/100      #Percentage Connectivity
margin = float(0.7)  #Margin for beta distribution centered at 0.5

NConnect = [[0 for col in range(N)] for row in range(N)]
NWeight = [[0 for col in range(2)] for row in range(N)]
for i in range(N):
    NWeight[i][0] = i + 1
    if i <= float(N)/2:
        NWeight[i][1] = 1
    else:
        NWeight[i][1] = -1

G=nx.erdos_renyi_graph(N,0)
pos=nx.circular_layout(G)

def make_edges(NConnect,NWeight):

    for i in range(N):
        x=NWeight[i][1]
        if x==1:
            G.node[i]['type']=0
        else:
            G.node[i]['type']=1

    for i in range(N):
        for j in range(N):
            if G.has_edge(i,j):
                G.remove_edge(i,j)

    for i in range(N):
        for j in range(N):
            x=NConnect[i][j]
            if x==1:
                G.add_edge(i,j)
    type0_node_list = [n for  (n,d) in G.nodes(data=True) if d['type'] == 0]
    type1_node_list = [n for  (n,d) in G.nodes(data=True) if d['type'] == 1]
```

```
    return type0_node_list,type1_node_list

def display_graph(type0_node_list, type1_node_list,edges):
    nx.draw_networkx_nodes(G,pos,
                           nodelist=type0_node_list,
                           node_color='r',
                           node_size=50)
    nx.draw_networkx_nodes(G,pos,
                           nodelist=type1_node_list,
                           node_color='b',
                           node_size=50)
    nx.draw_networkx_edges(G,pos,
                           edgelist=edges)
    plt.axis('off')
    plt.show()

### Function for adding a connection between two points ###
def AddCon(x, y, NConnect):
    if x == y:
        return NConnect
    else:
        NConnect[x][y] = 1
        NConnect[y][x] = 1
        return NConnect

### Function for deleting a connection between two points ###
def DelCon(x, y, NConnect):
    if x == y:
        return NConnect
    else:
        NConnect[x][y] = 0
        NConnect[y][x] = 0
        return NConnect


### Function to compare weights (True if the same) ###
def WeightCheckBool(i, j, NWeight):
    Bool = NWeight[i][1] == NWeight[j][1]
    return Bool

### Function to determine if a weight should change ###
def WeightCheck(NConnect, N, NWeight, x):
    global margin
    #Sum of connected weights for ratio
    a = 1 #1
    b = 1 #-1

    for i in range(N):
        if NConnect[x][i] == 1:
            w = NWeight[i][1]
            if w == 1:
```

```
                a = a+1
            else:
                b = b+1

    z = spst.beta.rvs(a,b,0,1)
    if z > (1+margin)*0.5:
        NWeight[x][1] = 1
    elif z < (1-margin)*0.5:
        NWeight[x][1] = -1

    return NWeight

### Function to determine if a connection should change ###
def ConCheck(NConnect, N, NWeight, x, y):

    r = float(sum(NConnect[x]))/(N-1)
    #print(r)
    Bool = NConnect[x][y] == 1 #True if connection else False
    Check = WeightCheckBool(x, y, NWeight)
    v = float(0.0)
    if Bool: #Connection is there
        if Check:
            if C >= (1-v)*r:
                if np.random.choice([True, False], p = [0.05, 0.95]):
                    NConnect = AddCon(x, y, NConnect)
            elif C <= (1+v)*r:
                if np.random.choice([True, False], p = [0.025, 0.975]):
                    NConnect = DelCon(x, y, NConnect)
        else:
            if C >= (1-v)*r:
                if np.random.choice([True, False], p = [0.025, 0.975]):
                    NConnect = AddCon(x, y, NConnect)
            elif C <= (1+v)*r:
                if np.random.choice([True, False], p = [0.05, 0.95]):
                    NConnect = DelCon(x, y, NConnect)

    else: #Connection is not there
        if C >= (1-v)*r:
            if Check:
                if np.random.choice([True, False], p = [0.05, 0.95]):
                    NConnect = AddCon(x, y, NConnect)
            else:
                if np.random.choice([True, False], p = [0.025, 0.975]):
                    NConnect = AddCon(x, y, NConnect)
        elif C <= (1+v)*r:
            if Check:
                if np.random.choice([True, False], p = [0.025, 0.975]):
                    NConnect = DelCon(x, y, NConnect)
            else:
                if np.random.choice([True, False], p = [0.05, 0.95]):
                    NConnect = DelCon(x, y, NConnect)
```

```
    return NConnect


def UpdateCon(NConnect, N, NWeight):
    for i in range(N):
        for j in range(N):
            NConnect = ConCheck(NConnect, N, NWeight, i, j)
    return NConnect


def UpdateWeight(NConnect, N, NWeight):
    for i in range(N):
        NWeight = WeightCheck(NConnect, N, NWeight, i)
    return NWeight



### Start program ###

#NConnect = InitMat(NConnect, N)

for i in range(100):
    #Initialise network with appropriate connections
    NConnect = UpdateCon(NConnect, N, NWeight)

type0_node_list,type1_node_list=make_edges(NConnect,NWeight)
fig1 = pl.gcf()
fig1.canvas.set_window_title('SimpleInitGraph')
display_graph(type0_node_list, type1_node_list,G.edges())

for i in range(N):
    print(NConnect[i][:])

print('\n')
for i in range(N):
    print(NWeight[i][:])

con=(nx.average_node_connectivity(G)/(N-1))
count = 0

while count < 50:
    count = count + 1
    con=(nx.average_node_connectivity(G)/(N-1))
    NConnect = UpdateCon(NConnect, N, NWeight)
    NWeight = UpdateWeight(NConnect, N, NWeight)
    type0_node_list,type1_node_list=make_edges(NConnect,NWeight)
    if count==25:
        fig2 = pl.gcf()
        fig2.canvas.set_window_title('SimpleMidGraph')
        display_graph(type0_node_list, type1_node_list,G.edges())
```

```
fig3 = pl.gcf()
fig3.canvas.set_window_title('SimpleFinGraph')
display_graph(type0_node_list, type1_node_list,G.edges())

print('\n')
for i in range(N):
    print(NConnect[i][:])
print('\n')
for i in range(N):
    print(NWeight[i][:])
print('\n' + str(count))

con=(nx.average_node_connectivity(G)/(N-1))*100
print("Average node connectivity is: " + str(con))
```

Social Network Python Code

```
import networkx as nx
import matplotlib.pyplot as plt
import random
import numpy
import collections
import pylab as pl
import itertools

N = int(input("Population:"))
n0=[]
n1=[0]
n2=[0]
w=numpy.ones(N)
a=0.5
b=0.5

NConnect = [[0 for col in range(N)] for row in range(N)]
NWeight = [[0 for col in range(2)] for row in range(N)]
def weight(a,b):
    for i in range(N):
        NWeight[i][0] = i
        NWeight[i][1] = numpy.random.choice([1, -1], p = [a, b])
    return NWeight

#Assign weights to nodes
NWeight=weight(a,b)
w2=dict(NWeight)

#Create the network
G = nx.MultiGraph()
E=nx.erdos_renyi_graph(N,1) #Fully Connected Network
#E=nx.erdos_renyi_graph(N,0.05) #Erdos-Renyi Network
#E=nx.scale_free_graph(N).to_undirected() #Scale-Free Network
G.add_nodes_from(E)
G.add_edges_from(E.edges())
H=nx.Graph()
```

```
H.add_path([N,N+1])
H.remove_edge(N,N+1)
G.add_nodes_from(H)
for i in range(N):
        G.add_edge(N,i)
        G.add_edge(N+1,i)

pos=nx.circular_layout(G)
labels={}
for i in range(N):
        labels[i]='$'+str(i)+'$'
labels[N]='$X$'
labels[N+1]='$Y$'

#Separate nodes into types
for n in G.nodes():
        G.node[n]['type'] = 0
type0_node_list = [n for  (n,d) in G.nodes(data=True) if d['type'] == 0]
type1_node_list = [n for  (n,d) in G.nodes(data=True) if d['type'] == 1]
type2_node_list = [n for  (n,d) in G.nodes(data=True) if d['type'] == 2]

edges=list(G.edges())
hub_edges=[]
for (u,v) in edges:
        if v==N or v==N+1:
                hub_edges.append((u,v))
counter=collections.Counter(hub_edges)

def newEdge(edges_n,w,w2,counter):
    pi=numpy.zeros(N)
    for (u,v) in edges_n:
        try:
            m= counter[(u,v)]
        except KeyError:
            m=0.
        pi[u]=m*w[u]
    cpi = numpy.cumsum(pi)
    #Chooses random edge to create with hub X (y=N) or Y (y=N+1)
    u=random.uniform(0,max(cpi))
    x=numpy.searchsorted(cpi, u)
    if w2[x]==1:
        y=N
    else:
        y=N+1
    newedge = (x,y)
    return newedge

def change_node(edge,type0_node_list):
        u=edge[0][0]
        v=edge[0][1]
        type0=u in type0_node_list
```

```
        #Join algorithm
        if ((u,v) in G.edges())==True and type0==True:
                if v==N:
                        G.remove_edge(u,v)
                        G.node[u]['type']=2
                else:
                        G.remove_edge(u,v)
                        G.node[u]['type']=1
        #Switch algorithm
        elif ((u,v) in G.edges())==False and type0==False:
                if v==N:
                        n=N+1
                        G.remove_edge(u,n)
                        G.add_edge(u,v)
                        G.node[u]['type']=1
                else:
                        n=N
                        G.remove_edge(u,n)
                        G.add_edge(u,v)
                        G.node[u]['type']=2
        #Return if neither algorithm can be used
        else:
                return


def display_graph(type0_node_list, type1_node_list, type2_node_list):
        #draw the type0 nodes (i.e. neutral individuals)
        nx.draw_networkx_nodes(G,pos,
                               nodelist=type0_node_list,
                               node_color='g',
                               node_size=200)
        #draw the hubs
        nx.draw_networkx_nodes(G,pos,
                               nodelist=[N],
                               node_color='r',
                               node_size=700)
        nx.draw_networkx_nodes(G,pos,
                               nodelist=[N+1],
                               node_color='b',
                               node_size=700)
        #draw the updated nodes of individuals
        nx.draw_networkx_nodes(G,pos,
                               nodelist=type1_node_list,
                               node_color='r',
                               node_size=200)
        nx.draw_networkx_nodes(G,pos,
                               nodelist=type2_node_list,
                               node_color='b',
                               node_size=200)
        #draw the edges and labels
        nx.draw_networkx_edges(G,pos,
```

```
                                edgelist=G.edges())
        nx.draw_networkx_labels(G,pos,labels,font_color='w',font_size=10)
        plt.axis('off')
        plt.show()

n0+=[len(type0_node_list)-2]
days=0
network=dict([(k,list(v.keys())) for k,v in  E.adjacency()])
#Show the initial network
display_graph(type0_node_list, type1_node_list, type2_node_list)
for i in range(100):
        new_edges=[]
        counter=collections.Counter(hub_edges)
        #Update the weights
        NWeight=weight(a,b)
        w2=dict(NWeight)
        for i in range(N):
                neighbours=network[i]
                edges_n=[]
                for j in range(len(neighbours)):
                        edges_n.append([item for item in hub_edges if neighbours[j] in item])
                edges_n = list(itertools.chain.from_iterable(edges_n))
                counter=collections.Counter(edges_n)
                new_edges.append(newEdge(edges_n,w,w2,counter))
        newedges=new_edges
        for (u,v) in newedges:
                type0_node_list = [n for  (n,d) in G.nodes(data=True) if d['type'] == 0]
                change_node([(u,v)],type0_node_list)
        n0+=[len(type0_node_list)-2]
        type1_node_list = [n for  (n,d) in G.nodes(data=True) if d['type'] == 1]
        n1+=[len(type1_node_list)]
        type2_node_list = [n for  (n,d) in G.nodes(data=True) if d['type'] == 2]
        n2+=[len(type2_node_list)]
        days += 1
#Show the final network
display_graph(type0_node_list, type1_node_list, type2_node_list)

#Plot a graph of time against number of individuals of each type
plt.plot(n0,'g-',label='type 0')
plt.plot(n1,'r-',label='type 1')
plt.plot(n2,'b-',label='type 2')
plt.xlabel("Time")
plt.ylabel("Number of Individuals")
plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
           ncol=3, mode="expand", borderaxespad=0.)
plt.show()
```

Network Segregation Python Code

```
import networkx as nx
import matplotlib.pyplot as plt
import random
import numpy
```

```
import collections
import pylab as pl
import itertools

#Population
N = int(input("Population:"))
#Aversion bias
p=float(input("Aversion Bias:"))
#Random graph with probability 0.05 of each node forming an edge
G=nx.erdos_renyi_graph(N,0.05)
E=len(G.edges())
#Assign random weights to each edge
for (u,v,d) in G.edges(data=True):
    d['weight'] = random.uniform(0,1)
labels={}
for i in range(N):
    labels[i]='$'+str(i)+'$'
    x=random.randint(0,1)
    if x==0:
        G.node[i]['type']=0
    else:
        G.node[i]['type']=1

#Separate nodes into their specific types
type0_node_list = [n for  (n,d) in G.nodes(data=True) if d['type'] == 0]
type1_node_list = [n for  (n,d) in G.nodes(data=True) if d['type'] == 1]
#Convert weights and edges to list form
weights=[d['weight'] for (u,v,d) in G.edges(data=True)]
edges=list(G.edges())

def display_graph(type0_node_list, type1_node_list,edges,weights):
    pos=nx.spring_layout(G,weight='weight')

    for i in range(1,len(pos)-1):
        for j in range(1):
            pos[i][j]=pos[i][j]*weights[i]

    nx.draw_networkx_nodes(G,pos,
                           nodelist=type0_node_list,
                           node_color='r',
                           node_size=50)
    nx.draw_networkx_nodes(G,pos,
                           nodelist=type1_node_list,
                           node_color='b',
                           node_size=50)
    nx.draw_networkx_edges(G,pos,
                           edgelist=edges,
                           length=weights)
    plt.axis('off')
    plt.show()
```

```
fig1 = pl.gcf()
fig1.canvas.set_window_title('InitialGraph')
display_graph(type0_node_list, type1_node_list, G.edges(data=True),weights)

network=dict([(k,list(v.keys())) for k,v in  G.adjacency()])

'''
Function for rewiring edges where the list of edges is an input.
Chooses a random edge to delete and then adds a new weighted edge
based to a different pair of nodes
'''
def rewire(edges):
    elarge=[(u,v) for (u,v,d) in G.edges(data=True) if d['weight'] >0.5]
    esmall=[(u,v) for (u,v,d) in G.edges(data=True) if d['weight'] <=0.5]
    r=random.randint(0,len(edges)-1)
    edge=edges[r]
    u=edge[0]
    neighbours_u=list(G.neighbors(u))
    v=edge[1]
    neighbours_v=list(G.neighbors(v))
    for (x,y,d) in G.edges(data=True):
        if (x,y)==(u,v):
            try:
                w=d['weight']
                break
            except KeyError:
                return
    #Probability of tie deletion
    pr=w*p
    #Tie deletion doesn't occur if this is met
    if random.uniform(0,1)<pr:
        return
    z=random.randint(0,1)
    for (x,y,d) in G.edges(data=True):
        if z==0 and x==u:
        #New node chosen from a list of nodes that aren't connected to u
            new=random.choice(list(G.nodes()^neighbours_u))
            G.remove_edge(u,v)
            G.add_weighted_edges_from([(u,new,random.uniform(3,4))])
            return
        elif z==1 and x==v:
        #New node chosen from a list of nodes that aren't connected to v
            new=random.choice(list(G.nodes()^neighbours_v))
            G.remove_edge(u,v)
            G.add_weighted_edges_from([(new,v,random.uniform(-3,-4))])
            return

#Rewires edges for 10000 iterations
for i in range(10000):
    edges=list(G.edges())
    rewire(edges)
```

```
    weights=[d['weight'] for (u,v,d) in G.edges(data=True)]

#Plots the final graph which should have reached the Markov stationary point
fig2 = pl.gcf()
fig2.canvas.set_window_title('FinalGraph')
display_graph(type0_node_list, type1_node_list,G.edges(data=True),weights)
```

Network Segregation MATLAB Code

```
clear;
K=4;
p=0.05;
q1=5*p/8; q2=(5-sqrt(5))*p/8; q3=(5+sqrt(5))*p/8;
C1=-3/100; C2=-(2*sqrt(5)+5)/50; C3=-(5-2*sqrt(5))/50;%random graph
%C1=1/50; C2=123/248; C3=61/15127; %short edges only
%C1=-9/50;C2=-451/1284;C3=183/3571; %long edges only
%The following equations are found by solving eq. (10)
e1=@(x) 12/25 + C1.*exp(-q1.*x) + C2.*exp(-q2.*x) + C3.*exp(-q3.*x);
e2=@(x) 6/25 + 3.*C1.*exp(-q1.*x)+0.5.*C2.*(1-sqrt(5)).*exp(-q2.*x)+...
    0.5.*C3.*(1+sqrt(5)).*exp(-q3.*x);
e3=@(x) 4/25 - 3.*C1.*exp(-q1.*x)+C2.*(2-sqrt(5)).*exp(-q2.*x)+...
    C3.*(2+sqrt(5)).*exp(-q3.*x);
e4=@(x)1-(12/25 + C1.*exp(-q1.*x) + C2.*exp(-q2.*x) + C3.*exp(-q3.*x))-...
    (6/25 + 3.*C1.*exp(-q1.*x)+0.5.*C2.*(1-sqrt(5)).*exp(-q2.*x)...
    +0.5.*C3.*(1+sqrt(5)).*exp(-q3.*x))-...
    (4/25 - 3.*C1.*exp(-q1.*x)+C2.*(2-sqrt(5)).*exp(-q2.*x)+...
    C3.*(2+sqrt(5)).*exp(-q3.*x));

hold on;
set(gca,'fontsize',20);
fplot(e1,[0 100],'LineWidth',2);
fplot(e2,[0 100],'LineWidth',2);
fplot(e3,[0 100],'LineWidth',2);
fplot(e4,[0 100],'LineWidth',2);
lgd=legend('0.25','0.5','0.75','1');
lgd.FontSize=20;
xlabel('x');
```