GEMF: GENERALIZED EPIDEMIC MODELING FRAMEWORK SOFTWARE IN PYTHON

HEMAN SHAKERI

Network Science and Engineering Group (NetSE)
Department of Electrical and Computer Engineering
Kansas State University
Manhattan, KS 66506-5204, USA

1. Examples in Python

In order to the determine nodal and edge-based transitions rates, we use node transition graph. We used NetworkX in our examples because of due to its ubiquitous use, although a user can consider other modules with minor alterations in the codes.

However, for individual-based epidemic models, transition graphs represent only the transition mechanism for each node in the network and not for the entire population. When defining an epidemic model, compartments should be defined first; For example, the SIS model has only two compartments: susceptible and infected.

Second, transitions between compartments, transition rates and their types should be defined. The inducer compartment and layers that define neighbor nodes must also be specified.

The following sections present examples of epidemic models that we simulated with Gemf. To implement the following sections user should import Gemf with the following line[1].

```
from GEMFPy import *
```

1.1. **SIS.** As mentioned, each node in an SIS model can be susceptible or infected; therefore, the number of compartments was denoted by $M = 2$. A susceptible node can become infected if it is surrounded by infected nodes. Infection process of a node with one infected neighbor is a Poisson process with transition rate β. The infection processes are stochastically independent of each other; therefore, for a susceptible node with more than one infected node in its neighborhood, the transition rate is the infection rate β times the number of infected neighbor nodes. The neighborhood of each node is determined by a contact network N. In addition to the infection process, a recovery process also exists. An infected node becomes susceptible again with a curing rate δ. The main characteristics and a node transition graph for the SIS model are shown in Table 1 and Figure 1.

Table 1. Descriptions of the SIS model

SIS					
State	Transition	Type	Parameter	Inducer	Layer
S	$(S \to E)$	edge-based	β	Neighbors in I	1
I	$(I \to S)$	node-based	δ		

Parameters in Table 1 can be entered by the following lines:

```
beta = 0.8; delta = 1;
Para = Para_SIS(delta,beta)
Net = NetCmbn([MyNet(G)])
```

where the function Para-SIS is defined as

[1]The latest version of Gemf can be found here.

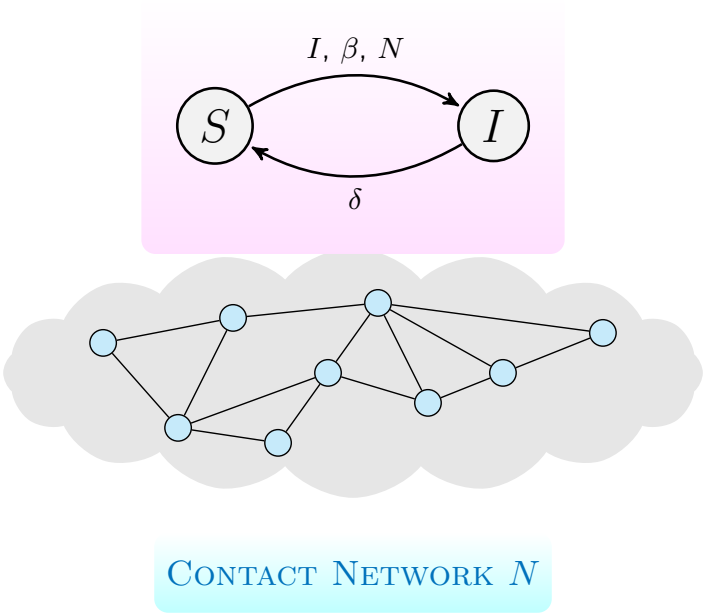

Figure 1. Schematic of the network-based SIS model

```
def Para_SIS(delta,beta):
    M = 2; q = np.array([1]); L = len(q);
    A_d = np.zeros((M,M)); A_d[1][0] = delta
    A_b = []
    for l in range(L):
        A_b.append(np.zeros((M,M)))
    A_b[0][0][1] = beta #[l][M][M]
    Para=[M,q,L,A_d,A_b]
    return Para
```

we can choose initial condition such that two nodes are initially in the first inducer compartment[2] and others are in the first compartment:

```
x0 = np.zeros(N)
x0 = Initial_Cond_Gen(N, Para[1][0], 2, x0)
```

1.1.1. *Simulation.* After defining Para for SIS model, we simulated an SIS model with $\beta = 1.2$ and $\delta = 1$, as shown in Figure 2. The simulation can be done by the following lines of codes: First define the duration of simulation

```
StopCond = ['RunTime', 20]
```

and finding the events:

```
ts, n_index, i_index, j_index = GEMF_SIM(Para, Net, x0, StopCond,n)
```

One output of the simulation can be the history for population of each compartment:

```
T, StateCount = Post_Population(x0, M, n, ts, i_index, j_index)
```

In Figure 2, the fraction of nodes in each state is shown.

[2]Python is using 0-based indexing.

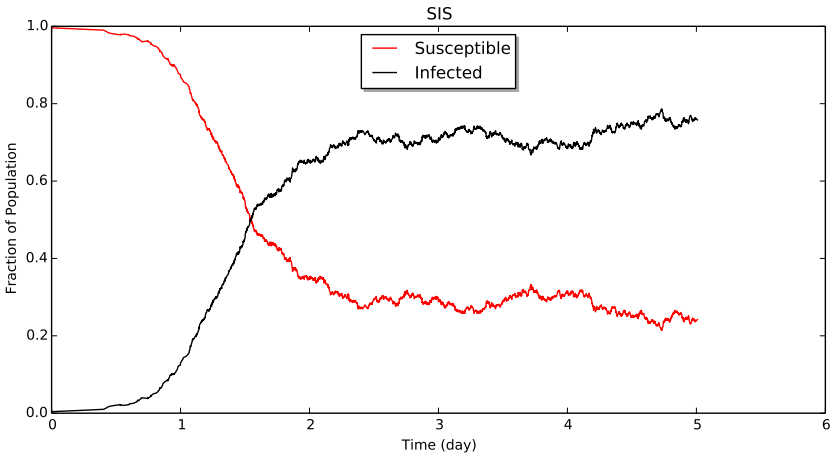

FIGURE 2. Simulation of the SIS model

Due to stochast nature of the simulation, we can repeat it for N times with the following line of codes with steps (see Section 2.4):

```
N = 2
T_final = 3
step = .1
Init_inf = 2
t_interval, f = MonteCarlo(StopCond, Init_inf, M, T_final, step, N, n)
```

where n is the entire population size, step is the chosen time step and f is the averaged population of each compartment in the time step.

1.2. **SIR.** In the Susceptible-Infected-Recovered (SIR) model, each node can be either susceptible, infected, or recovered (immune). Therefore, the number of compartments, denoted by M,in the SIR model, was $M = 3$. A susceptible node can become infected if it is surrounded by infected nodes. The infection process of a node with one infected neighbor is a Poisson process with transition rate β. Similar to SIS, infection processes are stochastically independent of each other. In addition to the infection process, a recovery process also exists. An infected node recovers and becomes immune with a recovery rate δ. The main characteristics and a node transition graph for the SIR model are shown in Table 2 and Figure 3.

TABLE 2. Descriptors of the SIR model

SIR multilayer					
State	Transition	Type	Parameter	Inducer	Layer
S	$(S \to E)$	edge-based	β	Neighbors in I	1
I	$(I \to R)$	node-based	δ		
R					

Parameters in Table 2 can be entered by the following lines:

```
beta = 1.2; delta = 1;
Para = Para_SIR(delta, beta)
Net = NetCmbn([MyNet(G)])
```

where the function Para-SIR is defined as

```
def Para_SIR(delta, beta):
    M = 3; q = np.array([1]); L = len(q);
    A_d = np.zeros((M,M));   A_d[1][2] = delta
    A_b = []
    for l in range(L):
        A_b.append(np.zeros((M,M)))
    A_b[0][0][1] = beta #[l][M][M]
    Para=[M,q,L,A_d,A_b]
    return Para
```

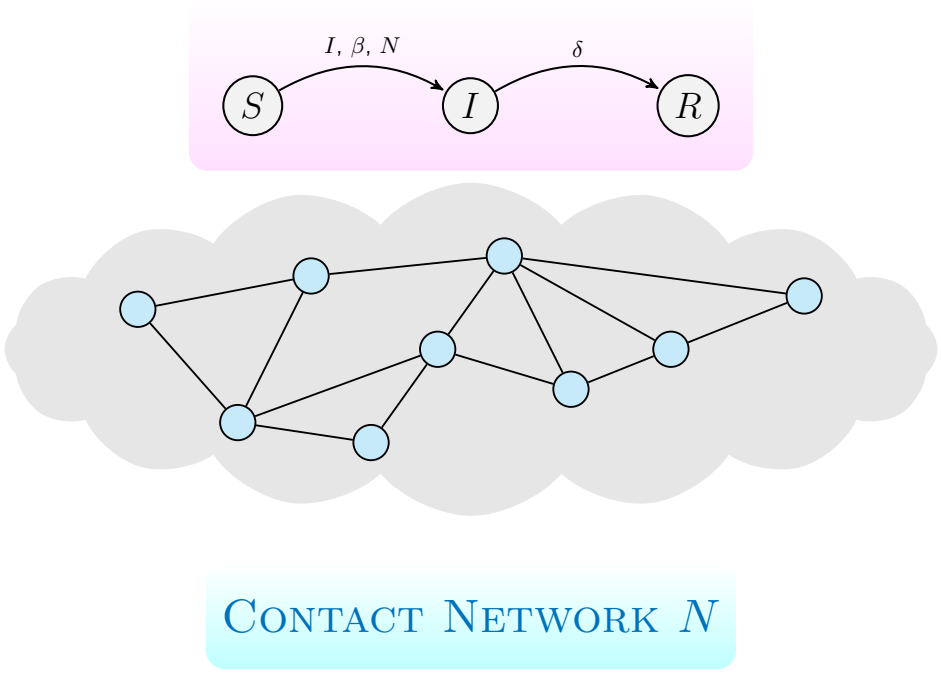

Figure 3. Node transition graph for the SIR model for nodes in N

1.2.1. *Simulation.* After defining Para for SIR model, we simulated an SIR model with $\beta = 1.2$, $\delta = 1$, as shown in Figure 3 for a Barabasi-Albert network with 500 nodes. Method is similar to SIS simulation in Section 1.1.1.

1.2.2. *SEIR.* In the Susceptible-Exposed-Infected-Recovered (SEIR) model, each node can be susceptible, exposed, infected, or recovered (immune). Therefore, $M = 4$. A susceptible node can become exposed, if it is surrounded by infected nodes. The infection process of a node with one infected neighbor is a Poisson process with transition rate β. The neighborhood of each node is determined by a contact network N. An exposed node is not yet infectious, but it will transition to the infected state with rate λ. Finally, an infected node recovers with a recovery rate δ. The main characteristics and a node transition graph for the SEIR model are shown in Table 3 and Figure 5.

Parameters in Table 3 can be entered by the following lines:

```
beta = 1.5; delta = 1; Lambda = .5
Para = Para_SEIR(delta, beta, Lambda)
Net = NetCmbn([MyNet(G)])
```

where the function Para-SEIR is defined as

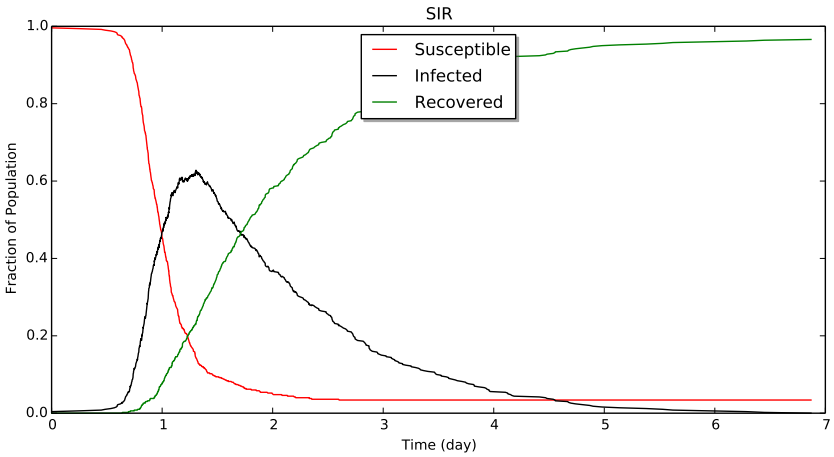

FIGURE 4. Simulation of the SIR model

TABLE 3. Descriptors of the SEIR

[h!]

SEIR multilayer					
State	Transition	Type	Parameter	Inducer	Layer
S	$(S \to E)$	edge-based	β	Neighbors in I	1
E	$(E \to I)$	node-based	λ		
I	$(I \to R)$	node-based	δ		

```
def Para_SEIR(delta, beta, Lambda):
    M = 4; q = np.array([2]); L = len(q);
    A_d = np.zeros((M,M));   A_d[1][2] = Lambda; A_d[2][3] = Lambda
    A_b = []
    for l in range(L):
        A_b.append(np.zeros((M,M)))
    A_b[0][0][1] = beta #[l][M][M]
    Para=[M,q,L,A_d,A_b]
    return Para
```

1.2.3. *Simulation.* After defining PARA for SEIR model, we simulated an SEIR model with $\beta = 1.2$, $\delta = 1$ and $\lambda = .4$, as shown in Figure 6.

1.3. **SAIS.** The Susceptible-Alert-Infected-Susceptible (SAIS) model was developed to incorporate individual reactions to the spread of a virus. In the SAIS model, each node (individual) can be susceptible, infected, or susceptible-alert. Therefore, the number of compartments in the SAIS model was $M = 3$. The recovery process is similar to recovery process in the SIS model, characterized by the recovery rate δ. The infection process of a susceptible agent is also similar to the infection process of the SIS model, determined by infection rate β and contact network N. However, in the SAIS model, a susceptible node can become alert if it senses infected agents in its neighborhood. The alerting transition rate is κ times the number of infected agents. An alert node can also become infected by a process similar to the infection process of a susceptible node. However, the infection rate for alert nodes is lower than susceptible nodes due to the adoption of preventive behaviors. The alert infection rate is denoted by β_a with $0 < \beta_a < \beta$. The main characteristics and a schematic for the SAIS model are shown in the following Table 4 and Figure 7.

Parameters in Table 4 can be entered by the following lines:

```
Para = Para_SAIS_Single(delta, beta, beta_a, kappa)
```

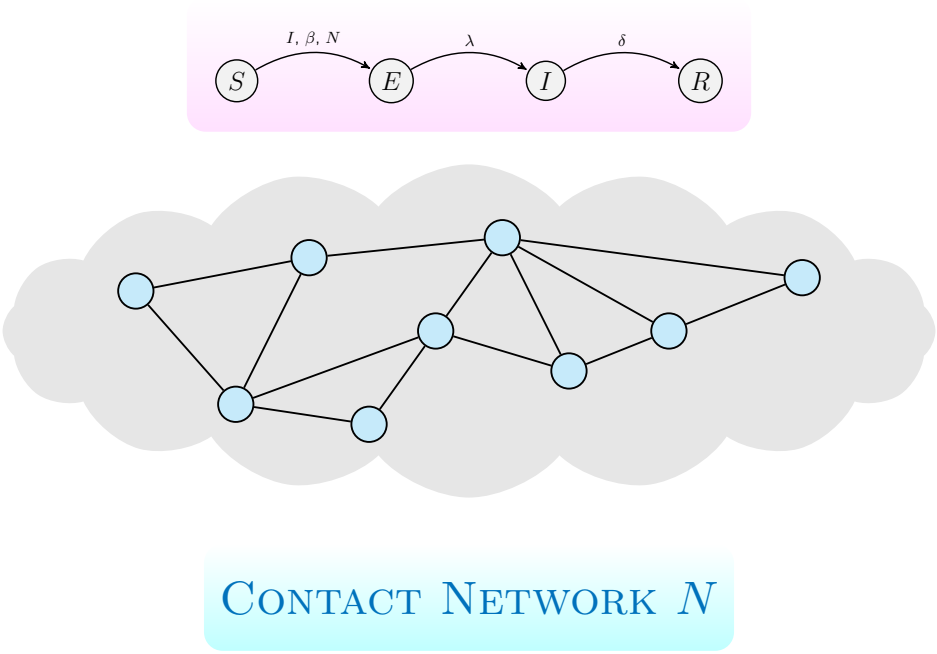

Figure 5. Node transition graph for the SEIR model for nodes in N

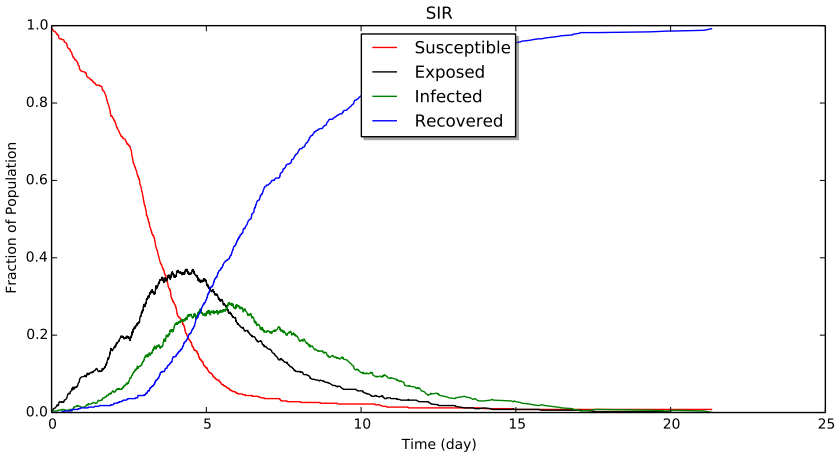

Figure 6. Simulation of the SEIR model

Table 4. Descriptors of the SAIS single layer model.

SAIS single Layer					
State	Transition	Type	Parameter	Inducer	Layer
S	$(S \to I)$	edge-based	β	Neighbors in I	1
	$(S \to A)$	edge-based	κ	Neighbors in I	1
I	$(I \to S)$	node-based	δ		
A	$(A \to I)$	edge-based	β_a	Neighbors in I	1

where the function Para-SAIS for single layer is defined as

```
def Para_SAIS_Single(delta, beta, beta_a, kappa):
    M = 3; q = np.array([1]); L = len(q);
    A_d = np.zeros((M,M)); A_d[1][0] = delta
    A_b = []
    for l in range(L):
        A_b.append(np.zeros((M,M)))
    A_b[0][0][1] = beta #[l][M][M]
    A_b[0][0][2] = kappa
    A_b[0][2][1] = beta_a
    Para = [M, q, L, A_d, A_b]
    return Para
```

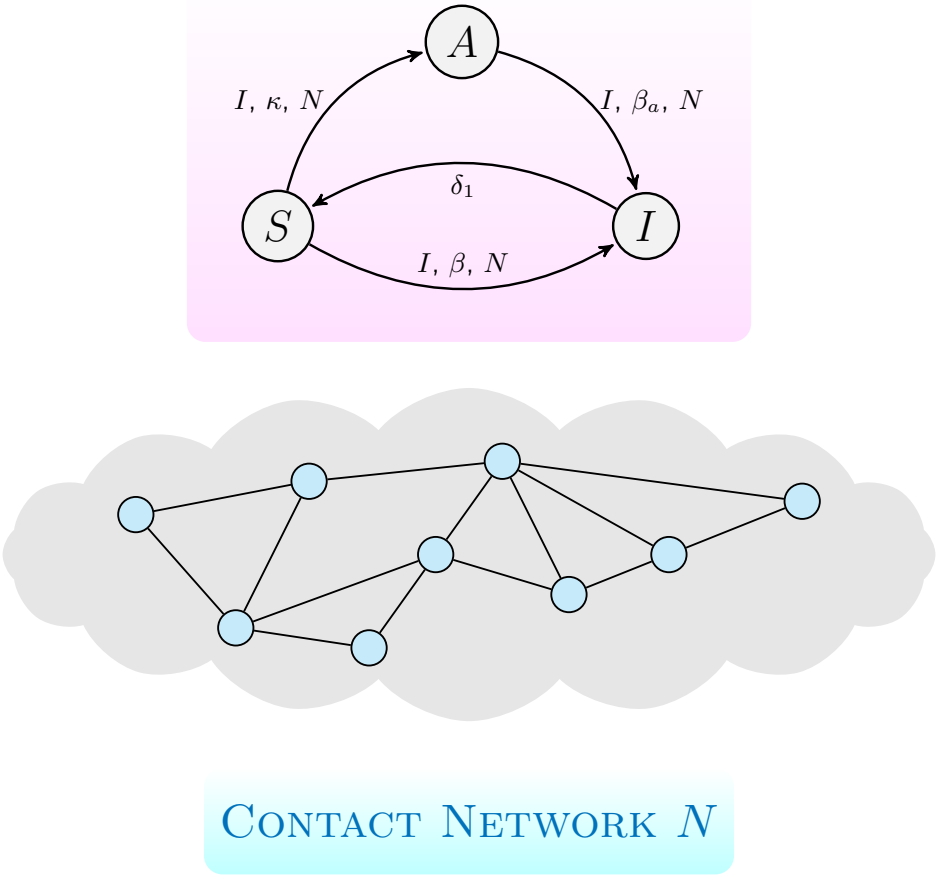

Figure 7. Node transition graph for the SAIS one layer model for nodes in N

1.3.1. *Simulation.* After defining Para for SAIS model, we simulated an SAIS model in one-layer (N), with $\beta = \frac{5}{\lambda_1(\mathcal{G}_1)}$, $\delta = 1$ and $\beta_a = \frac{0.5}{\lambda_1(\mathcal{G}_1)}$, and $\kappa = 0.2\beta$, as shown in Figure 8.

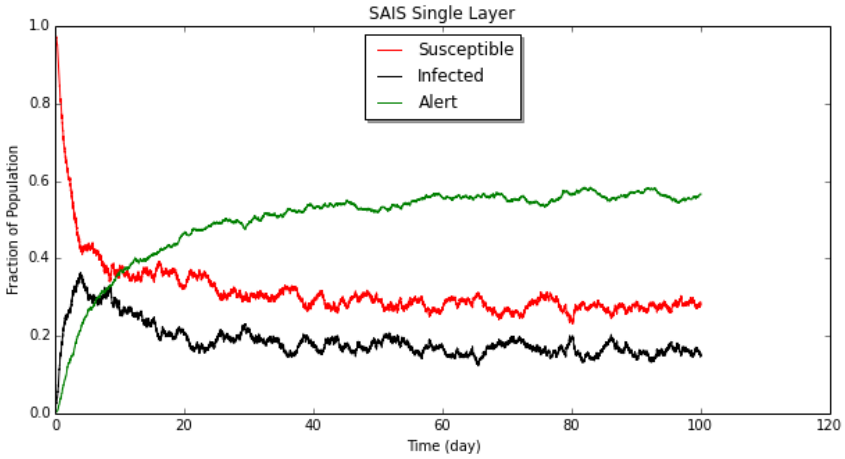

FIGURE 8. Simulation of the SAIS single layer model.

1.3.2. *SAIS Multilayer.* The SAIS model on a two layer network was developed to incorporate multiple sources of information to react to the spread of the virus. In the SAIS spreading model, each node (individual) can be either susceptible, infected, or susceptible-alert. Again, the number of compartments in the SAIS model was $M = 3$. The infection process of a susceptible agent was also similar to the infection process of the SIS model, determined by infection rate β and contact network N_A. However, in this version of the SAIS model, a susceptible node can become alert if it senses infected agents in its contact neighborhood or if it is notified about infected neighbors in an information network N_B. The alerting transition rate is κ times the number of infected agents in the contact network and μ times the number of infected agents in the notification network. An alert node can also become infected by a process similar to the infection process of a susceptible node. However, the infection rate for alert nodes β_a is lower than β due to the adoption of preventive behaviors such as using masks. The main characteristics and a schematic for the SAIS-2 layer model are shown in Table 5 and Figure 10.

TABLE 5. Descriptors of the SAIS two-layer model

SAIS multilayer					
State	Transition	Type	Parameter	Inducer	Layer
S	$(S \to I)$	edge-based	β	Neighbors in I	1
	$(S \to A)$	edge-based	κ	Neighbors in I	1
	$(S \to A)$	edge-based	μ	Neighbors in I	2
I	$(I \to S)$	node-based	δ		
A	$(A \to I)$	edge-based	β_a	Neighbors in I	1

Parameters in Table 5 can be entered by the following lines:

```
lambda1 = EIG1(G)[0]; delta = 1; beta = 5/lambda1; beta_a = .5/lambda1;
    kappa = .2*beta; mu = .5*beta
Para = Para_SAIS(delta, beta, beta_a, kappa, mu)
```

where the function Para-SAIS for two layer is defined as

```
def Para_SAIS(delta, beta, beta_a, kappa, mu):
    M = 3; q = np.array([1,1]); L = len(q);
    A_d = np.zeros((M,M)); A_d[1][0] = delta
    A_b = []
    for l in range(L):
```

```
        A_b.append(np.zeros((M,M)))
    A_b[0][0][1] = beta #[l][M][M]
    A_b[0][0][2] = kappa
    A_b[1][2][1] = beta_a
    A_b[1][0][2] = mu
    Para = [M, q, L, A_d, A_b]
    return Para
```

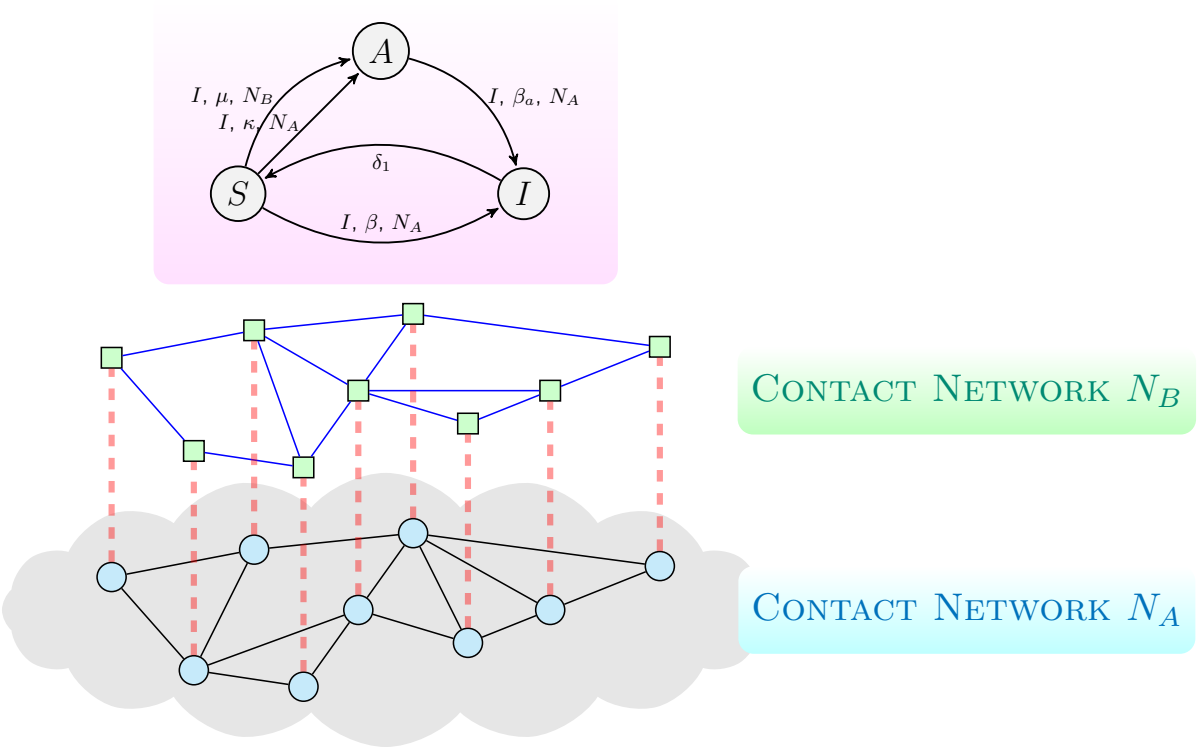

Figure 9

Figure 10. Node transition graph for the SAIS two-layer model on network with layers N_A and N_B.

1.3.3. *Simulation.* After defining Para for SAIS model, we simulated the process with $\beta = \frac{5}{\lambda_1(\mathcal{G}_1)}$, $\delta = 1$ and $\beta_a = \frac{0.5}{\lambda_1(\mathcal{G}_1)}$, $\kappa = 0.2\beta$, and $\mu = 0.5\beta$, as shown in Figure 11.

1.4. **Multiple interacting pathogen spreading SI_1SI_2S.** Assigning only one influencer compartment to one network layer allows different elegant analysis. However, a more general possibility is that an edge-based transition $m \to n$ occurs if a neighbor j, is in a subset of the compartments, such as $q_{l,1}$ or $q_{l,2}$. This case can be treated within the same structure, allowing the network layer to be counted twice. For example, we assumed that in the first layer the model had the influencer compartment $q_{l,1}$, and in the second layer, the graph has the influencer compartment $q_{l,2}$.

The SI_1SI_2S model is an extension of continuous-time SIS spreading of a single virus on a simple graph, to the modeling of competitive viruses on a two-layer network. In this model, each node is either susceptible, 1-infected, or 2-infected (i.e., infected by Virus 1 or 2, respectively). Virus 1 spreads through network N_1, virus 2 spreads through network N_2. In this competitive scenario, the two viruses are exclusive: a node

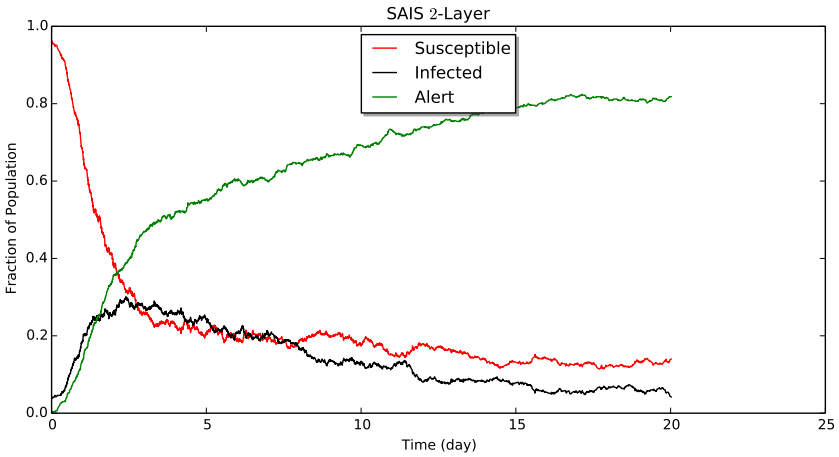

FIGURE 11. Simulation of the SAIS in 2 layer

cannot be infected by Virus 1 and Virus 2 simultaneously. Consistent with SIS propagation on a single layer, the infection and recovery processes for Virus 1 and 2 have similar characteristics. The curing process for 1-infected Node i is a Poisson process with recovery rate $\delta_1 > 0$. The infection process for susceptible Node i effectively occurs at rate $\beta_i Y_i(t)$, where $Y_i(t)$ is the number of 1-infected neighbors of node i at time t in layer N_1. Recovery and infection processes for Vvirus 2 are similarly described. The main characteristics and a node transition graph for the SI_1SI_2S model are shown in Table 6 and Figure 12.

TABLE 6. Descriptions of the SI_1SI_2S model. S: suscpetible, I_1: infected by virus 1, I_2: infected by virus 2,

SI_1SI_2S					
State	Transition	Type	Parameter	Inducer	Layer
S	$(S \to I_1)$	edge-based	β_1	Neighbors in I_2	1
	$(S \to I_2)$	edge-based	β_2	Neighbors in I_2	2
I_1	$(I_1 \to S)$	node-based	δ_1		
I_1	$(I_2 \to S)$	node-based	δ_2		

Parameters in Table 6 can be entered by the following lines in two different networks N1 (G) and N2 (H):

```
N = G.number_of_nodes()
lambda1_1 = EIG1(G)[0]; lambda1_2 = EIG1(H)[0]; delta1 = 1; beta1 = 5/
    lambda1_1;  delta2 = 1; beta2 = 5/lambda1_2;
Para = Para_SI1I2S(delta1, delta2, beta1, beta2)
Net = NetCmbn([MyNet(G), MyNet(H)])
x0 = np.zeros(N)
x0 = Initial_Cond_Gen(N, Para[1][0], 20, x0)
x0 = Initial_Cond_Gen(N, Para[1][1], 20, x0)
```

where the function Para-SI_1SI_2S is defined as

```
def Para_SI1I2S(delta1, delta2, beta1, beta2):
    M = 3; q = np.array([1,2]); L = len(q);
    A_d = np.zeros((M,M)); A_d[1][0] = delta1; A_d[2][0] = delta2
    A_b = []
    for l in range(L):
10
```

```
6           A_b.append(np.zeros((M,M)))
7       A_b[0][0][1] = beta1 #[l][M][M]
8       A_b[1][0][2] = beta2 #[l][M][M]
9
10      Para = [M, q, L, A_d, A_b]
11      return Para
```

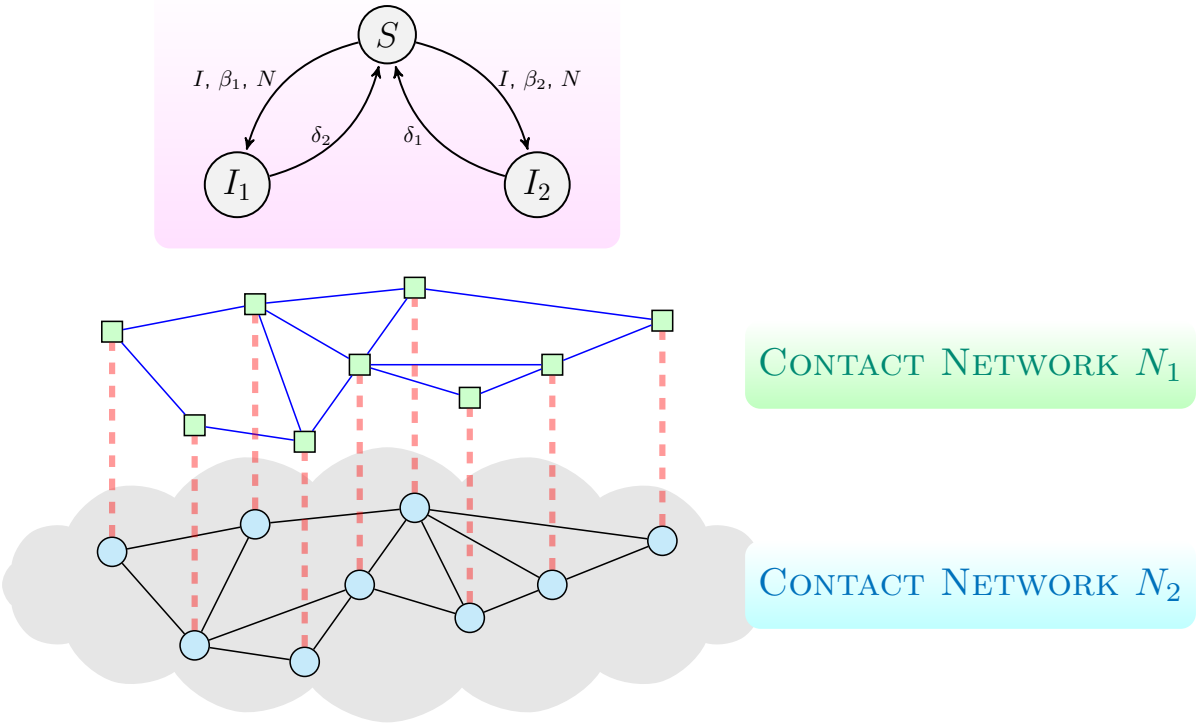

FIGURE 12. Node transition graph for the SI_1SI_2S in two layer model

1.4.1. *Simulation.* After defining PARA for this model, we simulate an S_1SI_2S model in one layer with $\beta_1 = \frac{5}{\lambda_1(\mathcal{G}_1)}$, $\beta_2 = \frac{5}{\lambda_1(\mathcal{H}_1)}$, $\delta_1 = 1$ and $\delta_2 = 1$ in Figure 13.

2. AN OVERVIEW OF THE FUNCTIONS

In the subsequent sections, we describe the following functions of GEMF:

(1) Initialization functions
   (a) NET
   (b) NETCOMB
   (c) PARA
   (d) INITIALCOND
(2) Simulations
   (a) SIM
   (b) POSTPROCESS
   (c) MONTECARLO
(3) Output
   (a) VISUALIZATION

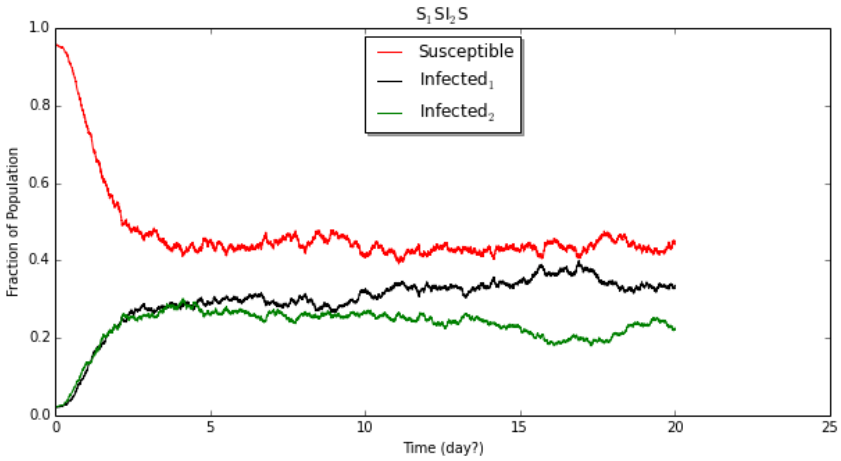

FIGURE 13. Simulation of the S_1SI_2S model

2.1. **Initialization.**

2.1.1. *"NET": Converting network data.* GEMF converts graph information into graph adjacency list format with function NET; therefore, we recorded i, j, and w in vectors L_2, L_1 and W for each edge (i, j, w) . L_2 are neighbors of L_1 with weight W, and we sorted L_2 and re-arranged L_1 and W with resulting sorted arguments in order to organize these data. All network data was acquired with L_1, L_2 and W.

The NEIGHBORHOODDATA function was used, which takes L_1 and L_2 as its inputs and returns 5 vectors outputs. *Neighvec* and *NeighWeight* are vectors of neighbors, and their weights I_1 and I_2, respectively, are node indices and d is their edge multiplicity. Benefits of this representation are described in Section 2.2.

We defined NNEIGHBORHOODDATA in order to distinguish between neighbors of node i with nodes that have i as neighbors. This function, is useful when we are dealing with a directed network and has the same structure as the previous function. Outputs of this function, *NNeighvec* and *NNeighWeight*, are vectors of adjacent nodes (not neighbors) and their edge weights respectively and NI_1 and NI_2 are indices and Nd is edge multiplicity. Function NET returns all above information for a single layer.

2.1.2. *Combining network layers data.* For each layer, we obtained the required information from NET, and we combined them with the NETCOMB function.

$$\begin{aligned} &\text{NETCOMB}\left(\{Net_1, \cdots, Net_L\}\right) = \\ &\qquad \left[\left[Net_1\left(1\right), \cdots, Net_L\left(1\right)\right], \cdots, \left[Net_1\left(H\right), \cdots, Net_L\left(H\right)\right]\right]_{1\times L} \end{aligned} \tag{1}$$

where $Net_l = \text{NET}\left(\mathcal{G}_l\right)$.

2.1.3. *Transition rates.* We used PARA function to enter the required data for transition rates, as described in Section **??**. A nodal transition rate matrix is an $M \times M$ matrix in which entry mn represents the rate of nodal transition $m \to n$:

$$A_\delta \triangleq \left[\delta_{mn}\right]_{M\times M}. \tag{2}$$

An edge-based transition rate matrix, corresponding to the network layer l, is an $M \times M$ matrix in which entry mn represents the rate $\beta_{l,mn} > 0$ of edge-based transition $m \to n$:

$$A_\beta \triangleq \left[\left[\beta_{1,mn}\right]_{M\times M}, \cdots, \left[\beta_{L,mn}\right]_{M\times M}\right]_{1\times L} \tag{3}$$

Examples of how to use this functions are presented in Section 1.

2.1.4. *Initial condition.* With "INITIALCONDGEN" function the initial status of each individual in the population can be determined and various approaches can be used to do this.

- User input: Initial condition is directly chosen by the user.
- Fixed initial infected population: N_J individuals randomly chosen to be in compartment J.

2.2. **Simulations.** GEMF uses an event-driven approach to simulate the stochastic process. This method is advantageous compared to the discretized method. For example, in discretization approach, no transition may occur in several time increments dt or several transition may occur in one time increment; therefore, computation time for the event-based method is not unnecessarily longer and on the other side the solution is more accurate and captures more events compared to the discretized method (See [**?**], [**?**]).

Number of neighbors in influencer compartment N_q. As discussed in Section **??**, one of the key factors in edge-based transitions is the number of neighbors in influencer compartment, N_q. N_q is an $L \times N$ array, representing the number of influencer compartment for each node in each layer, weighted by edge weights. Because node status changes in each event, N_q is updated after each event. From Section 2.1.4, initial status of all nodes $X^0_{M\times N}$ is obtained. For example, if $X[:,4]^T = \begin{bmatrix}0 & 1 & \cdots & 0\end{bmatrix}_{1\times M}$, then node 4 is in compartment 2.

To compute N_q, GEMF goes over all nodes in each layer. Using network data from NETCOMB, all neighbors of node n in layer l can be derived via

$$N_{ln} = Neigh[l][I_1[l,n] : I_2[l,n]] \tag{4}$$

with weights:

$$W_{ln} = NeighWeight[l][I_1[l,n] : I_2[l,n]]. \tag{5}$$

Using (5), entries of N_q (influencer neighbors) can be determined by

$$N_q[l,n] = \sum_{i=1}^{|N_{ln}|} X[q[l], N_{ln}[i]] \cdot W_{ln}[i] \tag{6}$$

where $|N_{ln}|$ is the cardinality of set N_{ln}.

2.2.1. *Rate of changes.* From Section 2.1.3, we entered A_β and A_δ through PARA. The simulation code initially generated b_{il}, which is an arrays:

$$b_{il} \triangleq \begin{bmatrix} \begin{bmatrix} \sum_{i=1}^{M}\beta_{1,1i} \\ \vdots \\ \sum_{i=1}^{M}\beta_{1,Mi} \end{bmatrix}_{M\times 1} & \cdots & \begin{bmatrix} \sum_{i=1}^{M}\beta_{L,1i} \\ \vdots \\ \sum_{i=1}^{M}\beta_{L,Mi} \end{bmatrix}_{M\times 1} \end{bmatrix}_{1\times L} \tag{7}$$

where b_{il} represents the sum of edge-based transition rates of each compartment in each layer.

The array of edge-based transition rates matrix for each compartment in all layers b_i was

$$b_i \triangleq \begin{bmatrix} \begin{bmatrix} \beta_{1,11} & \cdots & \beta_{L,11} \\ \beta_{1,12} & \cdots & \beta_{L,12} \\ \vdots & \ddots & \vdots \\ \beta_{1,1M} & \cdots & \beta_{L,1M} \end{bmatrix}_{M\times L} & \cdots & \begin{bmatrix} \beta_{1,21} & \cdots & \beta_{L,21} \\ \beta_{1,22} & \cdots & \beta_{L,22} \\ \vdots & \ddots & \vdots \\ \beta_{1,2M} & \cdots & \beta_{L,2M} \end{bmatrix}_{M\times L} \end{bmatrix}_{1\times M} \tag{8}$$

For each compartment, the total leaving rate due to nodal transition was derived from 2 (by summing up each row of matrix A_δ):

$$d_i = \begin{bmatrix} \sum_{i=1}^{M}\delta_{1i} \\ \vdots \\ \sum_{i=1}^{M}\delta_{Mi} \end{bmatrix}_{M\times 1}. \tag{9}$$

2.2.2. *Total Rates.* Using d_i and b_{il}, total transition rates for each node were generated as

$$R_{in} = (d_i \mathbf{1}_{1\times N})_{M\times N} \circ X + (b_{il} N_q)_{M\times N} \circ X \tag{10}$$

where $\circ$ represents element-wise multiplication.

In order to find the total rate of change for the entire system, we re-added the rates. For example, for the total rate of change for each compartment in the entire network, we introduce R_i:

$$R_i = \begin{bmatrix} \sum_{i=1}^{N} R_{in}[1,i] \\ \vdots \\ \sum_{i=1}^{N} R_{in}[M,i] \end{bmatrix}_{M\times 1} \tag{11}$$

and for the total rate of change for the entire system, we introduced R:

$$R = \sum_{i=1}^{M} R_i[i]\,. \tag{12}$$

2.2.3. *Updating system status after an event.* The initial state for all nodes was generated according to Section 2.1.4. Because all random processes are Poisson processes, the assumption was made that the next event would occur in time δt:

$$\delta t = \frac{-\ln(\mathsf{rand})}{R} \tag{13}$$

where $0 \le \mathsf{rand} \le 1$ is a generated random number. During this event one of the nodes changes its status. We determined which compartment changed by drawing a sample among M compartments with probability distribution R_i; this compartment was called i_s.

Once the leaving compartment was identified, we wanted to know which node experienced the transition. Therefore, we drew a sample from N nodes with probability distribution $R_i n[i_s,:]$ (i.e., i_s row of matrix $R_i n$) and called this Node n_s.

To find the new status (compartment) of Node n_s, again Gemf randomly draws the new compartment j_s among M compartments with the following probability distribution:

$$p_{j_s}^T = A_\delta[i_s,:]^T + b_i[i_s] N_q[:,n_s]\,. \tag{14}$$

Drawing samples with given probability distribution is done with RndDraw function.

With δt, i_s, j_s, and n_s, Gemf had all necessary information to update the network status and apply required changes with the occurred event. However, Gemf had to update X matrix and the future rate of transitions.

Because Node n_s changed its status from i_s to j_s, we have:

$$X[i_s,n_s] = 0,\ X[j_s,n_s] = 1. \tag{15}$$

To update R_i, we subtracted the column in R_{in} that corresponded to Node n_s (i.e., $R_{in}[:,ns]$) and then we updated

$$R_{in}[:,n_s] = d_i \circ X[:,n_s] + (b_i N_q[:,n_s]) \circ X[:,n_s]\,. \tag{16}$$

Now we add $R_{in}[:,n_s]$ to R_i. Next if any of the old or new compartment are in influencer category in any layer, code should update N_q matrix. First, we find neighbors of node n_s:

$$N_{ln} = Neigh[l][I_1[l][n_s] : I_2[l][n_s]] \tag{17}$$

$$WeightedNeigh = NeighW[l][I_1[l][n_s] : I_2[l][n_s]] \tag{18}$$

(19)

Then we conducted the following steps for all these neighbors:

- If the old compartment i_s was an influencer compartment in layer l, we do the following removed n_s as their infected neighbors and recorded the weight of the edge. We also updated R_{in}. For n, the k'th neighbor of n_s was

$$N_q[l][n] -= NNeighW[l][NI_1[l][n_s]+k] \tag{20}$$

$$R_{in}[:,n] -= NNeighW[l][NI_1[l][n_s]+k]\,(b_{il}[:,n] \circ X[:,n]) \tag{21}$$

  where $-=$ indicates subtracting to current value.
- If the new compartment j_s was an influencer compartment in layer l, we added n_s as their infected neighbors and recorded the weight of the edge. We also updated R_{in}. For n, the kth neighbor of n_s was

$$N_q[l][n] += NNeighW[l][NI_1[l][n_s]+k] \tag{22}$$

$$R_{in}[:,n] += NNeighW[l][NI_1[l][n_s]+k]\,(b_{il}[:,n] \circ X[:,n]) \tag{23}$$

  where $+=$ indicates adding to current value.

We stacked n_s, j_s, and i_s into n_{index}, j_{index}, and i_{index}, respectively, and then we recalculated R_i and R and prepared for the next event.

2.3. **Post processing.** From Sim, we obtained the set of time increments of occurring events, t_s. The cumulative sum of t_s, T, was the time history of events. *StateCount*, an $M \times (|t_s|+1)$ array conveying the total number of nodes in each compartment in each time step, , was also generated. The First column of *StateCount* is initial condition:

$$StateCount[:,1] = \sum_{i=1}^{N} X_0[:,i]\,. \tag{24}$$

For the remainder of *StateCount*, PostPopulation generated a temporary array $DX_{M\times 1}$ in each event, with the following non-zero elements:

$$DX[i_{index}[k]] = -1 \tag{25}$$

$$DX[j_{index}[k]] = 1, \tag{26}$$

and zero on the other elements. In each event, we obtained the following recursion:

$$StateCount[:,k+1] = StateCount[:,k] + DX \tag{27}$$

PostPopulation returns T and *StateCount*.

2.4. **Monte carlo simulation.** In order to obtain a reliable result for stochastic simulation, it is necessary to repeat random processes had to be repeated for many times and the results need to be averaged.

In an event-based analysis, the number of events are not identicale for different simulations; therefore, arrays that show the state of the group in each simulation will not be of the same size and they cannot be added and averaged.

In order to average several random processes, a ubiquitous time interval with a desired time increment must be defined.

Therefore, the function MonteCarlo, uses histogram counting. For all simulations, it finds the closest previous event for the time increments and then maps these events on the new time interval. With the new unified time scale, the average of all processes was derived.

[1] Network Science and Engineering Group, Department of Electrical and Computer Engineering, Rathbone Hall, Kansas State University, Manhattan, KS 66506, USA

E-mail address: heman@ksu.edu