



Esame IA: Intelligent Models

Relazione Progetto:

Epidemic Super Spreader in Networks

Studente: Ludovico Guercio

Matricola: 340036

DIPARTIMENTO DI MATEMATICA E INFORMATICA,
UNIVERSITÀ DEGLI STUDI DI PERUGIA

Indice

1	Obbiettivo	1
2	SIR Model	1
2.1	Implementazione Sistema	1
2.2	Nodi Super-Spreader	2
3	Codice del programma	2
3.1	Modulo Sir.py	3
3.1.1	Metodo costruttore	3
3.1.2	Getters	4
3.1.3	Metodo draw_network	5
3.1.4	Metodo della simulazione	6
3.2	Individuazione Nodi Super-Spreader	8
3.2.1	Metodo centrality_metrics	10
3.2.2	Metodo super_spreader	11
3.2.3	Nodi spreader nell'heatmap	11
3.3	Modulo SimulazioneSIR.ipynb	12
4	Simulazione epidemica	13
4.1	Facebook network	14
4.1.1	Facebook super-spreaders	17
4.2	Twitch network	19
4.2.1	Twitch super-spreaders	23

1 Obbiettivo

L'obbiettivo della traccia del progetto è quello di implementare un sistema di "epidemic diffusion" simulandolo in una rete complessa per poi determinare i nodi "**super spreader**".

2 SIR Model

Il sistema consiste in una variazione basata sul modello SIR (Susceptible - Infected - Recovered). In questi modelli infatti, data una popolazione, si va a simulare la diffusione di una epidemia (ad esempio di una malattia) durante un periodo di tempo. Ad ogni periodo di tempo, la diffusione è calcolata in base ai dei parametri.

In una rete inizialmente tutti nodi sono nello stato di "Susceptible" e possono successivamente diventare "Infected", e poi "Recovered"; il cambiamento di stato è condizionato dai parametri di definizione e condizione dell'epidemia.

2.1 Implementazione Sistema

Il sistema implementato lavora nel seguente modo: dato un file csv, che descrive i nodi e gli archi della rete, si va a generare una network iniziale; questa, poi, viene elaborata per eseguire successivamente la simulazione epidemico.

L'elaborazione iniziale permette di impostare lo stato di partenza della rete per la simulazione.

Inizialmente tutti i nodi vengono segnati nello stato di "Susceptibile" e di "Infected"; naturalmente, una parte dei nodi della rete vengono settati infetti permettendo di eseguire la simulazione. I nodi infetti iniziali sono definiti tramite un numero totale di tale nodi e assegnati casualmente.

Il passaggio di stato di un nodo è definito come segue:

Modello SIRS

- **Susceptible – > Infected**
- **Infected – > Recovered**
- **Recovered – > Susceptible**

Come anticipato, i passaggi di stato dei nodi sono dipendenti da diversi parametri:

- **p_trans**: indica la probabilità di transizione della malattia da un nodo **Infected** ad un altro **Susceptible**. Più la probabilità è alta, più un nodo che è connesso ha un altro infetto, ha più probabilità di contrarre la malattia

- **t_rec**: indica il tempo di recupero dalla malattia; una volta passato questo tempo, il nodo torna nello stato di **Recovered**.
- **t_sus**: indica il tempo per cui un nodo **Recovered** torni allo stato di **Susceptible**; una volta tornato in questo stato, il nodo non è più immune alla malattia
- **t_sim**: indicata il tempo totale della simulazione del sistema. Si utilizzano valori **integer** simulando che essi rappresentano i giorni

Modello SIR

- **Susceptible – > Infected**
- **Infected – > Recovered**

Il sistema permette di analizzare anche il caso speciale per cui un nodo non torni allo stato di "Susceptible", quindi rimanendo immune per sempre alla malattia. In tal caso si lavorerà con il parametro **t_sus** tendente all'infinito:

2.2 Nodi Super-Spreader

Il programma permette di poter individuare i nodi "Super Spreader" di una determinata rete attraverso una serie di simulazioni randomiche.

I nodi **super spreader** sono quei nodi che hanno la maggiore capacità e tendenza a comunicare con più nodi. Nel caso di tale sistema, si intende la facilità per cui un nodo riesce facilmente a trasmettere la malattia ad altri nodi; solitamente questi nodi sono anche considerati come nodi "hub", mentre il resto di nodi hanno capacità drasticamente minori di trasmissione.

L'individuazione di questi nodi super spreader, posto come obiettivo finale, viene fatto tramite un metodo preciso implementato nella classe **SIR** del modulo python **Sir.py**. Questo metodo analizza le misure di centralità della rete confrontando i loro valori anche attraverso l'uso di mappe di classificazione disponibili grazie alla libreria python **matplotlib**.

3 Codice del programma

Il programma è struttura principalmente su 2 file python:

- **Sir.py**
- **simulationSIR.pynb**

Nel modulo `Sir.py` viene implementata una classe denominata **Sir**, la quale descrive le caratteristiche e funzionalità per inizializzare e simulare una rete.

Il secondo file è un notebook python utilizzabile, ad esempio, in Google Colab per un migliore uso di librerie grafiche come *NetworkX* (per i grafi e reti), *matplotlib* (per grafici, plotting ecc.)

3.1 Modulo Sir.py

Rappresenta il modulo python in cui viene definito il funzionamento vero e proprio del sistema SIR. Al suo interno è definita una classe **Sir**.

La classe **Sir** descrive quell'oggetto che permette di inizializzare una rete per il modello simulativo, visualizzare le caratteristiche e metriche, effettuare simulazioni.

3.1.1 Metodo costruttore

```
class Sir:  
  
    def __init__(self, init_infect, filename, delimiter=' '): #init infect = numero int di nodi infetti  
        df = pd.read_csv('content/' + filename, delimiter) #delimiter di default = ','  
        self.graph = nx.from_pandas_edgelist(df, source='from', target='to')  
        self.init_infect = init_infect  
        self.init_state = self.set_init_infect()
```

Il metodo `__init__` permette di inizializzare il grafo della rete per il modello simulativo. Vengono passati come parametri:

- `init_infect`: numero di nodi iniziali infetti
- `file`: il nome del file csv della rete
- `delimiter`: separatore dei valori nel file csv; di default è settato come uno spazio

Dati i parametri, si apre il file csv e si interpreta i dati al suo interno che descrivono gli archi dei nodi; quindi viene poi trasformato il csv in un grafo attraverso *NetworkX*. Infine si applica il metodo `set_init_infect()` per settare i nodi inizialmente infetti.

```

def set_init_infect(self):
    #get nodes
    nodes = self.graph.nodes()
    sample = rd.sample(nodes, self.init_infect) #assegno tot nodi casuali come infetti
    color_map = []
    #COLORI STATE
    CELESTE = '#abcdef'
    ROSSO = '#ff5349'
    #set parametri iniziali
    for node in nodes:
        #set initial infected
        if node in sample:
            self.graph.nodes[node]['State'] = 'I'
            self.graph.nodes[node]['T_rec'] = 0
            self.graph.nodes[node]['T_sus'] = 0
            color_map.append(ROSSO) #colore rosso
        else:
            self.graph.nodes[node]['State'] = 'S'
            self.graph.nodes[node]['T_rec'] = 0
            self.graph.nodes[node]['T_sus'] = 0
            color_map.append(CELESTE) # colore celeste

    self.draw_network()

```

`set_init_infect()` assegna randomicamente i nodi infetti iniziali sul totale posto come parametro iniziale. Per far ciò, per ogni nodo nell'array `sample` contenente i nodi da infettare, modifica gli attributi di tali nodi del grafo assegnandogli lo stato di "I" di infetto, `T_rec` e `T_sus` uguale a 0. Per il resto dei nodi del grafo, invece, gli viene attribuito lo stato di "S" suscettibile e con tempo di recupero e di suscettibilità sempre uguali a 0. A livello grafico, i nodi Infetti sono colorati di **rosso**, i nodi Suscettibili di **celeste**, mentre i nodi Guariti di **verde**.

Si richiama la funzione `draw_network()` per stampare la rete a video con gli stati aggiornati dopo l'inizializzazione.

Ad esempio, già subito con questi due metodi, possiamo definire un oggetto SIR che rappresenta una nuova rete inizializzata con i parametri di nodi infetti iniziali:

```

network = Sir(40, 'network.csv')
#oppure
network = Sir(40, 'network.csv', delimiter = ',', )

```

3.1.2 Getters

All'interno della classe ho definito dei metodi **getters**, utili per poter ritornare le informazioni e metriche relative alla rete e agli stati dei loro nodi:

- `getNeighbors`: dato un nodo, ritorna una lista di tutti i suoi nodi vicini
- `getNeighbors_Sus`: dato un nodo, ritorna una lista di tutti i suoi nodi vicini che hanno lo stato di "Suscettibile"
- `getSusceptible`: ritorna la lista di tutti i nodi nello stato di "Suscettibile"

- `getInfected`: ritorna la lista di tutti i nodi nello stato di "Infetto"
- `getRecovered`: ritorna la lista di tutti i nodi nello stato di "Guariti"

```

def getNeighbors(self, node):
    return [n for n in self.graph.neighbors(node)]


def getNeighbors_Susceptibile(self, node):
    neigh_susceptible = []
    for n in self.graph.neighbors(node):
        if self.graph.nodes[n]['State'] == 'S':
            neigh_susceptible.append(n)

    return neigh_susceptible


def getSusceptible(self):
    susceptible = []
    for n in self.graph.nodes:
        if self.graph.nodes[n]['State'] == 'S':
            susceptible.append(n)

    return susceptible


def getInfected(self):
    infected = []
    for n in self.graph.nodes:
        if self.graph.nodes[n]['State'] == 'I':
            infected.append(n)

    return infected


def getRecovered(self):
    recovered = []
    for n in self.graph.nodes:
        if self.graph.nodes[n]['State'] == 'R':
            recovered.append(n)

    return recovered

```

3.1.3 Metodo `draw_network`

Questa funzione d'utilità permette, dato un grafo di una rete, di riportare a video il grafo della rete con tutti i stati dei nodi. Richiamando i metodi `getters`, si recuperano la liste dei nodi suddivisi per i loro stati, che vengono poi colorati secondo i loro colori.

Si utilizza per la stampa i metodi della libreria di **NetworkX**, `nx.spring_layout()`, `nx.draw_networkx_nodes()`, `nx.draw_networkx_edges()`

```
#draw the network with node states
def draw_network(self):
    color_map = []
    susceptible = self.getSusceptible()
    infected = self.getInfected()
    recovered = self.getRecovered()

    for s in susceptible:
        color_map.append('■#abcdef') # colore celeste

    for i in infected:
        color_map.append('■#ff5349') #colore rosso

    for r in recovered:
        color_map.append('■#98ff98') #colore verde

    plt.figure(figsize=(20,10))
    pos = nx.spring_layout(self.graph, iterations = 15, seed=1721)
    nx.draw_networkx_nodes(self.graph, pos, node_size=100, node_color=color_map)
    nx.draw_networkx_edges(self.graph, pos, alpha=0.2)
    plt.axis('off')
    plt.show()
```

3.1.4 Metodo della simulazione

Il metodo **simulation** è quello che permette di eseguire la simulazione nella nostra rete. **Prende come input il grafo della rete e i parametri personalizzabili di simulazione.** Come descritto in precedenza nelle generalità del progetto, questi parametri sono:

- **p_trans**: indica la probabilità di transizione della malattia da un nodo **Infected** ad uno **Susceptible**
- **t_rec**: indica il tempo di recupero dalla malattia da parte di un nodo **Recovered**
- **t_sus**: indica il tempo per cui un nodo **Recovered** torni allo stato di **Susceptible**
- **t_sim**: indicata il tempo totale della simulazione del sistema.

Esempio utilizzo del metodo di simulazione:

```
simulazione = network.simulation(0.05, 7, 20, 60)

# dove p_trans = 0.05 , t_rec = 7, t_sus = 20, t_sim = 60
```

La simulazione all'interno della rete avviene come segue:

conoscendo il tempo totale di simulazione, per ogni intervallo di questo, quindi i giorni, si verifica il cambiamento di stato da parte dei nodi: ad ogni istante di tempo t, si recuperano tutti i nodi con i loro attributi di stato e parametri di tempo:

```
def simulation(self, p_trans, t_rec, t_sus, t_sim):

    G = self.graph
    #Initial/live/final data list per grafici e statistiche
    tot_S = []
    tot_I = []
    tot_R = []
    Timeline = [] #array temporale per grafico e statistiche
    color_map = []
    #COLORI STATE
    CELESTE = '#abcdef'
    ROSSO = '#ff5349'
    VERDE = '#98ff98'

    for t in range(1, t_sim):
        Timeline.append(t)
        #update list node
        susceptible = self.getSusceptible()
        infected = self.getInfected()
        recovered = self.getRecovered()

        s_upd = len(susceptible)
        i_upd = len(infected)
        r_upd = len(recovered)

        tot_S.append(len(susceptible))
        tot_I.append(len(infected))
        tot_R.append(len(recovered))
```

La simulazione della diffusione della malattia viene effettuata tramite operazioni e controlli per ogni tipo di nodo:

- **Infezione:** per ogni nodo nello stato di **Infected** si recuperato tutti i suoi nodi vicini suscettibili grazie al metodo `getNeighbors_Sus`; questi nodi essendo vicini degli infetti, possono contrarre la malattia. Se `p_trans` è maggiore di una possibile probabilità di contatto tra i due nodi, definita randomicamente, **il nodo suscettibile vicino passa allo stato di infected**
- **Guarigione:** Per ogni nodo nello stato di **Infected**, se il suo `t_rec` è maggiore o uguale al tempo `t_rec` impostato per la simulazione, allora il nodo è guarito, quindi **passa allo stato di recovered**; altrimenti il suo `t_rec` aumenta di 1
- **Vulnerabilità:** per ogni nodo nello stato di **Recovered**, passato un tempo `t_sus` maggiore o uguale di `t_sus` impostato per la simulazione, il nodo torna ad essere vulnerabile alla malattia, quindi **ritorna allo stato di susceptible**; altrimenti il suo `t_sus` aumenta di 1

```

for i in infected:
    neighbors = self.getNeighbors_Susceptibile(i)
    for n in neighbors:
        #set random probabilit for that node
        p = np.random.rand()
        if p <= p_trans:
            G.nodes[n]['State'] = 'I'
            G.nodes[n]['T_rec'] = 0
            G.nodes[n]['T_sus'] = 0
            color_map.append(ROSSO) #colore rosso

#guarigione
for j in infected:
    if G.nodes[j]['T_rec'] >= t_rec:
        G.nodes[j]['state'] = 'R'
        G.nodes[j]['T_sus'] = 0
        G.nodes[j]['T_rec'] = 0
        color_map.append(VERDE) #colore verde
    else:
        G.nodes[j]['T_rec'] += 1

#ritornare ad essere suscettibili
for r in recovered:
    if G.nodes[r]['T_sus'] >= t_sus:
        G.nodes[r]['State'] = 'S'
        G.nodes[r]['T_rec'] = 0
        G.nodes[r]['T_sus'] = 0
        color_map.append(CELESTE) # colore celeste
    else:
        G.nodes[r]['T_sus'] += 1

plt.plot(Timeline, tot_S, label="SIR Susceptible")
plt.plot(Timeline, tot_I, label="SIR Infected")
plt.plot(Timeline, tot_R, label="SIR Recovered")
plt.title('SIR Simulation')
plt.show()
print(f"Giorno {t} :\nSuscettibili: {s_upd}\nInfetti: {i_upd}\nGuariti: {r_upd}")
self.draw_network()
#return tot_S, tot_I, tot_R

```

La funzione termina con la definizione di due output: il primo output è un grafico creato attraverso `matplotlib` che riporta le metriche dell'andamento e variazione degli stati dei nodi durante il tempo simulativo.

L'altro output corrisponde al disegno grafico delle reti sociali una volta terminata la simulazione, questo grazie alla funzione `draw_network()`.

3.2 Individuazione Nodi Super-Spreader

I nodi **super-spreader** hanno una cruciale importanza all'interno di reti complesse, soprattutto quando si analizzano trasmissione di informazioni all'interno della rete, come nel contesto del progetto di diffusione epidemica.

L'obiettivo è quello di implementare una tecnica per individuare questi nodi nella rete. Attraverso la ricerca e letture di alcuni articoli e documentazioni (<https://www.frontiersin.org/articles/10.3389/fphy.2022.955727/full>, <https://www.nature.com/articles/>

`srep27823`), è possibile studiare questi nodi attraverso gli algoritmi di analisi delle misure di centralità

La soluzione implementata consiste nell'utilizzo di cinque algoritmi di misura della centralità dei nodi, adatti per modelli simulativi epidemici di grandi e complesse reti (anche rete sociali):

- **Betweness Centrality:** misura l'importanza di un nodo all'interno di una shortest-path. Più questo nodo è presente in più shortest-path, maggiore è il suo valore di centralità. La misura è molto importante nei sistemi di simulazioni epidemiche.
Dato un nodo V , si calcola il numero P di shortest-path, tra i e j che lo attraversano; si calcola poi il totale T di shortest-path e poi si calcola il rapporto tra P e T ; si esegue tutto per ogni coppia di nodi i e j.
Il costo computazionale è di: $O(|V||E|)$.
- **Closeness Centrality:** misura le shortest per ogni coppia di nodi nella rete. Rapresenta una misura di velocità di trasmissione da un nodo all'altro; un alto valore, indica che l'informazione (o in questo caso la malattia) arriva più velocemente in tutta la rete. Questa misura è molto importante nei sistemi di simulazione epidemiche. Si calcola con il reciproco della lunghezza media delle shortest-path da un nodo verso tutti gli altri.
Il costo computazionale è di
- **Eigenvector Centrality:** misura la centralità tra un nodo e i suoi vicini; nodi influenti sono quelli vicini a nodi con valore di eigenvector elevato. Nodi acquisiscono importanza se sono vicini ad altri nodi importanti, aumentando quindi la probabilità e velocità di trasmissione. L'algoritmo è simile a quello che Google utilizza per le pagine di ricerca (PageRank). La misura è importante in sistemi di simulazione epidemiche
- **Degree Centrality:** misura la centralità di un nodi in base al numero dei suoi nodi vicini. In reti di larga scala rimane sempre un fattore importante da non sottovalutare, in quanto si può avere molti nodi con molti vicini, di conseguenza avere nodi influenti in tutta la densità della rete

Come plus, ho deciso anche di prendere in analisi la **VoteRank Centrality**: è misura di rilevata importanza, il suo funzionamento è anche esso simile a PageRank di Google. Nell'algoritmo ogni nodo, per ogni turno, vota il suo nodo spreader influente, il nodo che riceve più voti per ogni turno diventa quindi un nodo-spreader; in base al numero di voti, si realizza il rank dei nodi super-spreader. La capacità di voto dei nodi vicini al nodo spreader diminuisce ogni turno successivo. L'algoritmo è ritenuto molto efficiente su reti di larga scala.

L'individuazione, poi, è supportata anche graficamente attraverso l'utilizzo di **heatmap** evidenziando l'importanza di questi nodi direttamente nel grafo della rete.

3.2.1 Metodo centrality_metrics

La funzione permette di calcolare le misure di centralità prese in analisi. La libreria di **NetworkX** implementa direttamente i metodi degli algoritmi di misura di centralità, così semplificando direttamente l'uso diretto:

```
def centrality_metrics(self):

    G = self.graph
    dc = nx.degree_centrality(G)
    cc = nx.closeness_centrality(G)
    bc = nx.betweenness_centrality(G)
    ec = nx.eigenvector_centrality(G)
    vr = nx.voterank(G, 15) #primi 15 più votati

    #from dict to pandas df
    dc_df = pd.DataFrame.from_dict({
        'node': list(dc.keys()),
        'degree centrality': list(dc.values())
    })

    cc_df = pd.DataFrame.from_dict({
        'node': list(cc.keys()),
        'closeness centrality': list(cc.values())
    })

    bc_df = pd.DataFrame.from_dict({
        'node': list(bc.keys()),
        'betweenness centrality': list(bc.values())
    })

    ec_df = pd.DataFrame.from_dict({
        'node': list(ec.keys()),
        'eigenvector centrality': list(ec.values())
    })

    vr_df = pd.DataFrame({'node':vr})
```

Tutti i valori vengono raccolti in dizionari che poi convertiti in dataframe di pandas per essere poi utilizzati nei i metodi `super_spreader()` `heatmap_centrality()`

3.2.2 Metodo super_spreader

```
def super_spreader(self):

    #get df centrality metrics
    dc, cc, bc, ec, VR = self.centrality_metrics()
    #sort top 10 and combine dataframe
    ec_sorted = ec.sort_values('eigenvector centrality', ascending=False)
    ec_sorted.index = np.arange(1, len(ec_sorted) + 1)

    cc_sorted = cc.sort_values('closeness centrality', ascending=False)
    cc_sorted.index = np.arange(1, len(cc_sorted) + 1)

    bc_sorted = bc.sort_values('betweenness centrality', ascending=False)
    bc_sorted.index = np.arange(1, len(bc_sorted) + 1)

    dc_sorted = dc.sort_values('degree centrality', ascending=False)
    dc_sorted.index = np.arange(1, len(dc_sorted) + 1)
    #voterank già ordina per importanza valore(rank), quindi setto solo gli index
    VR.index = np.arange(1, len(VR) + 1)

    BC = bc_sorted['node'][:15]
    CC = cc_sorted['node'][:15]
    EC = ec_sorted['node'][:15]
    DC = dc_sorted['node'][:15]

    ss = pd.merge(BC, CC, left_index=True, right_index=True)
    ss = pd.merge(ss, EC, left_index=True, right_index=True)
    ss = pd.merge(ss, DC, left_index=True, right_index=True)
    ss = pd.merge(ss, VR, left_index=True, right_index=True)
    ss.columns = ['BC', 'CC', 'EC', 'DC', 'VR']
    print(tabulate(ss, headers='keys', tablefmt='psql'))
```

La funzione permette di raccogliere i migliori nodi super-spreader influenti nella rete. Utilizza il metodo `centrality_metrics()` per raccogliere tutti i valori delle metriche: Infine viene realizzata una tabella con i migliori 15 nodi, individuati da ogni misura.

3.2.3 Nodi spreader nell'heatmap

La misura di centralità tramite heatmap è un ottima tecnica di supporto dove è possibile andare ad individuare gli **"hotspot"** della rete, zone in cui vengono segnalati i nodi come potenziali super-spreader.

La funzione implementata, permette, data la struttura della rete e l'algoritmo di misura di centralità del nodo, di evidenziare nel grafo tutti i nodi colorati secondo una scala di colore che segue parte dai valori minimi fino ai massimi di quella misura di centralità. Ad esempio, come nell'attuale implementazione, si utilizza una scala di colore di calore "heat": quindi nodi con meno rilevanza avranno un colore più scuro, mentre i nodi man mano che sono più rilevanti avranno sempre un colore più caldo.

Per l'output grafico si utilizza i metodi di plotting della libreria `matplotlib` di Python. Inoltre raccogliendo i dati delle misure di centralità, vengono poi stampati in output anche i migliori 15 nodi di ogni misura con i loro relativi valori.

```

def heatmap_centrality(self, metrics_name):

    G = self.graph
    cm = {}
    #get metrics
    if metrics_name == 'degree centrality':
        cm = nx.degree_centrality(G)
    if metrics_name == 'closeness centrality':
        cm = nx.closeness_centrality(G)
    if metrics_name == 'betweenness centrality':
        cm = nx.betweenness_centrality(G)
    if metrics_name == 'eigenvector centrality':
        cm = nx.eigenvector_centrality(G)

    df = pd.DataFrame.from_dict({
        'node': list(cm.keys()),
        metrics_name: list(cm.values())
    })
    sort_metrics = df.sort_values(metrics_name, ascending=False)
    top_metrics = sort_metrics[:15]

    plt.figure(figsize=(16,8))
    pos = nx.spring_layout(G, iterations = 15, seed=1721)
    nodes = nx.draw_networkx_nodes(G, pos, node_size=100, cmap=plt.cm.plasma,
                                   node_color=list(cm.values()),
                                   nodelist=cm.keys())
    nodes.set_norm(mcolors.SymLogNorm(linthresh=0.01, linscale=1, base=10))
    edges = nx.draw_networkx_edges(G, pos, alpha=0.2)

    plt.title(metrics_name)
    plt.colorbar(nodes)
    plt.axis('off')
    plt.show()
    print("Top 15:")
    print(top_metrics)

```

3.3 Modulo SimulazioneSIR.ipynb

Il file python notebook `simulazioneSIR.ipynb` viene implementato in Google Colab per poter simulare e visualizzare al meglio gli output grafici della rete e delle sue statistiche.

Il notebook permette di importare:

- il modulo `networkSIR.py`: descrive la classe **Sir** che permette di effettuare la simulazione
- i file csv: contengono i nodi e archi delle reti

SimulationSIR.ipynb

```

** IMPORT DEL MODULO PYTHON **

from google.colab import files
src = list(files.upload().values())[0]
open('NetworkSIR.py', 'wb').write(src)

```

```

** IMPORT FILE CSV **

uploaded = files.upload()

** IMPORT DELLA CLASSE SIR **

from NetworkSIR import SIR

```

Una volta importata la classe, è possibile andare a creare un nuovo oggetto **Sir**, inizializzare la rete ed effettuare le simulazioni epidemiche.

4 Simulazione epidemica

Per verificare ed analizzare il funzionamento del sistema implementato, ho deciso di utilizzare due big network prese direttamente da <https://snap.stanford.edu/data/>:

- **ego-Facebook:**
 - facebook_combined.csv (4039 nodi)
- **twitch-gamers:**
 - musae_ENGB.edges.csv (6960 nodi)

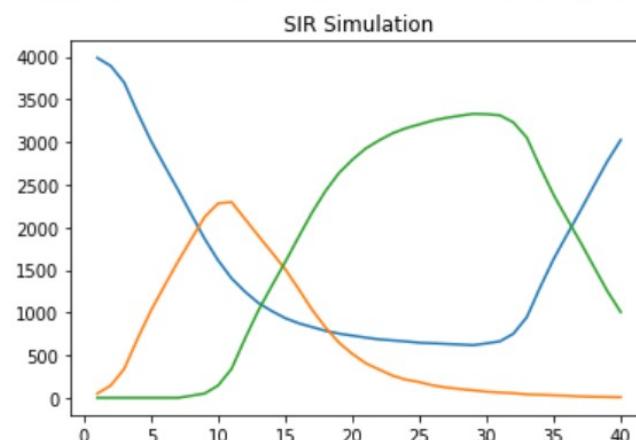
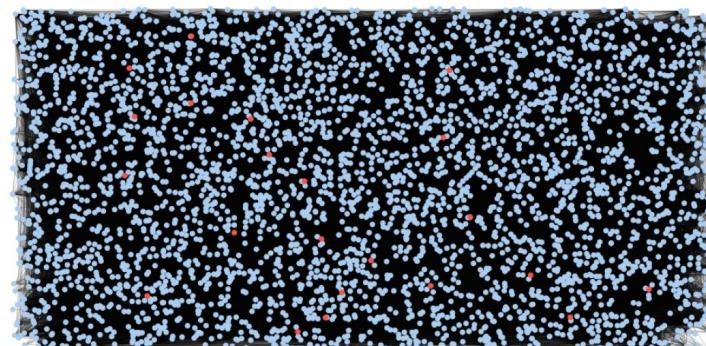
La prima rete può essere considerata una "base" large network, in cui in numero di 4000 nodi permettono di distinguersi ed essere ritenuta una rete grande, ma non di grande complessità; mentre la seconda rete, con quasi 7000 nodi, risulta una large network con un importante dimensione.

Date queste due largest network, si vanno ad effettuare delle simulazioni random andando a modificare i vari parametri di input.

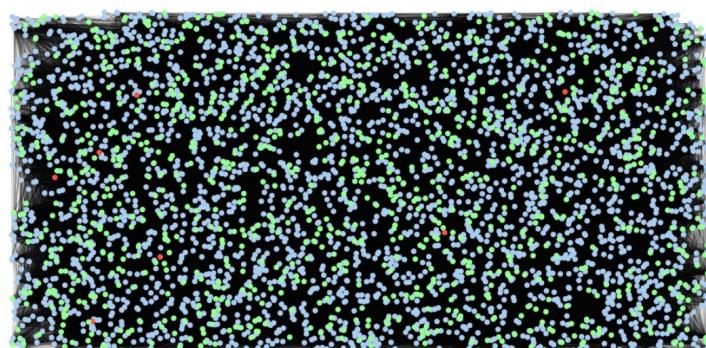
4.1 Facebook network

Risultati di alcune simulazioni:

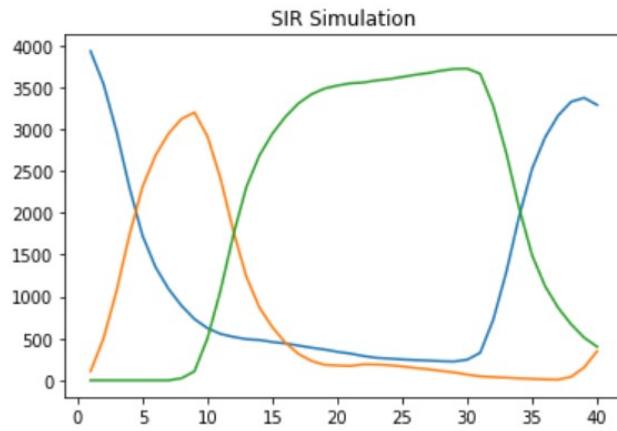
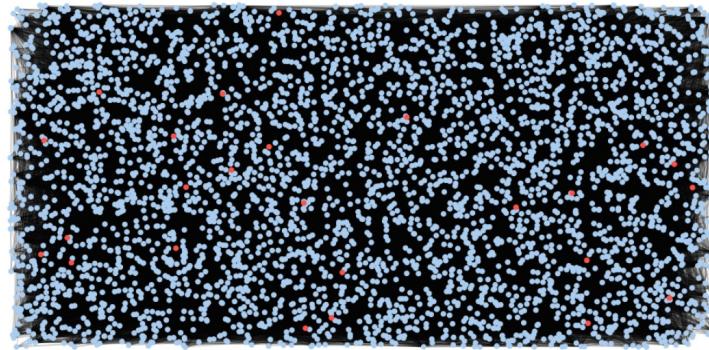
```
#Simulazione con probabilita trasmissione al 2%  
  
facebook1 = Sir(25, 'facebook_combined.csv')  
facebook1.simulation(0.02, 7, 21, 40) #input parametri simulazione:  
    p_trans, t_rec, t_sus, t_sim
```



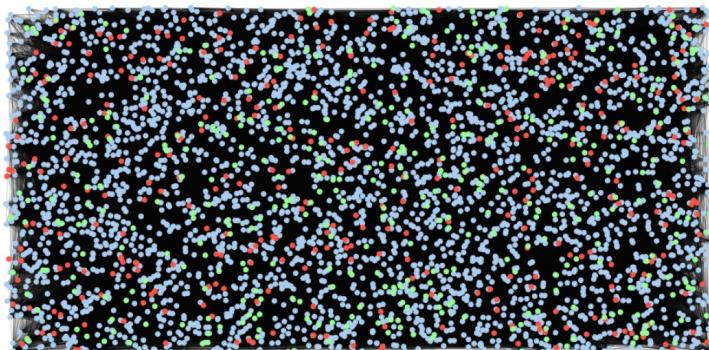
Giorno 40 :
Suscettibili: 3025
Infetti: 8
Guariti: 1006



```
#Seconda simulazione con probabilit infezione al 5%  
  
facebook2 = Sir(25, 'facebook_combined.csv')  
facebook2.simulation(0.05, 7, 21, 40) #input parametri simulazione:  
    p_trans, t_rec, t_sus, t_sim
```



Giorno 40 :
 Suscettibili: 3289
 Infetti: 348
 Guariti: 402



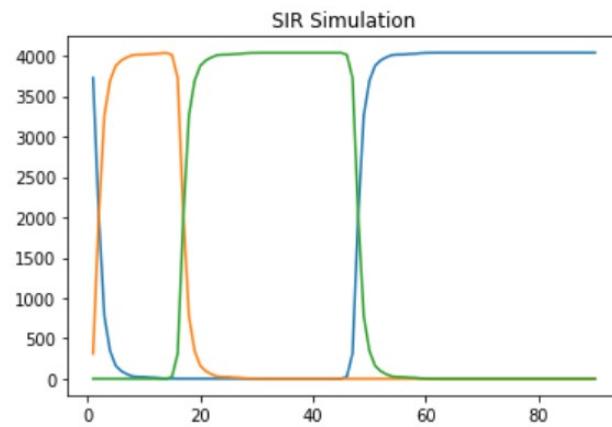
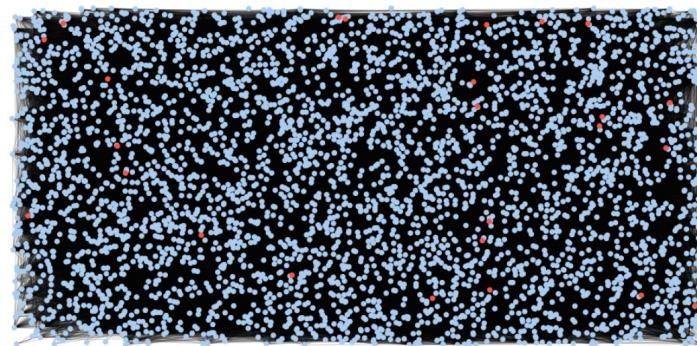
Le due simulazioni avvengono con lo stesso parametro iniziale di nodi infetti, ma questi ultimi sono assegnati sempre randomicamente.

L'andamento dell'epidemia è condizionata da i parametri iniziali della simulazione e dalla "influenza" dei nodi inizialmente infetti.

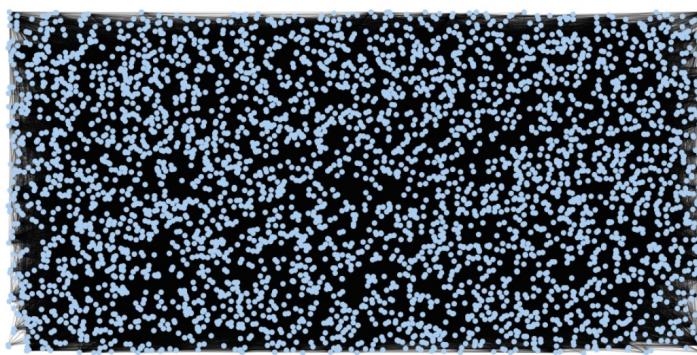
Quello che si può più notare è che con l'aumentare della probabilità di trasmissione (`p_trans`) il picco degli infetti nei primi tempi iniziali è maggiore, quindi si ha inizialmente un maggiore spread di infezione.

```
#aumento p_trans a 25%, t_rec 14 gg, t_sus 30 gg in periodo di 90 gg.

fb5 = Sir(25, 'facebook_combined.csv')
fb5.simulation(0.25, 14, 30, 90) #input parametri simulazione:p_trans,
# t_rec, t_sus, t_sim
```



Giorno 90 :
 Suscettibili: 4039
 Infetti: 0
 Guariti: 0



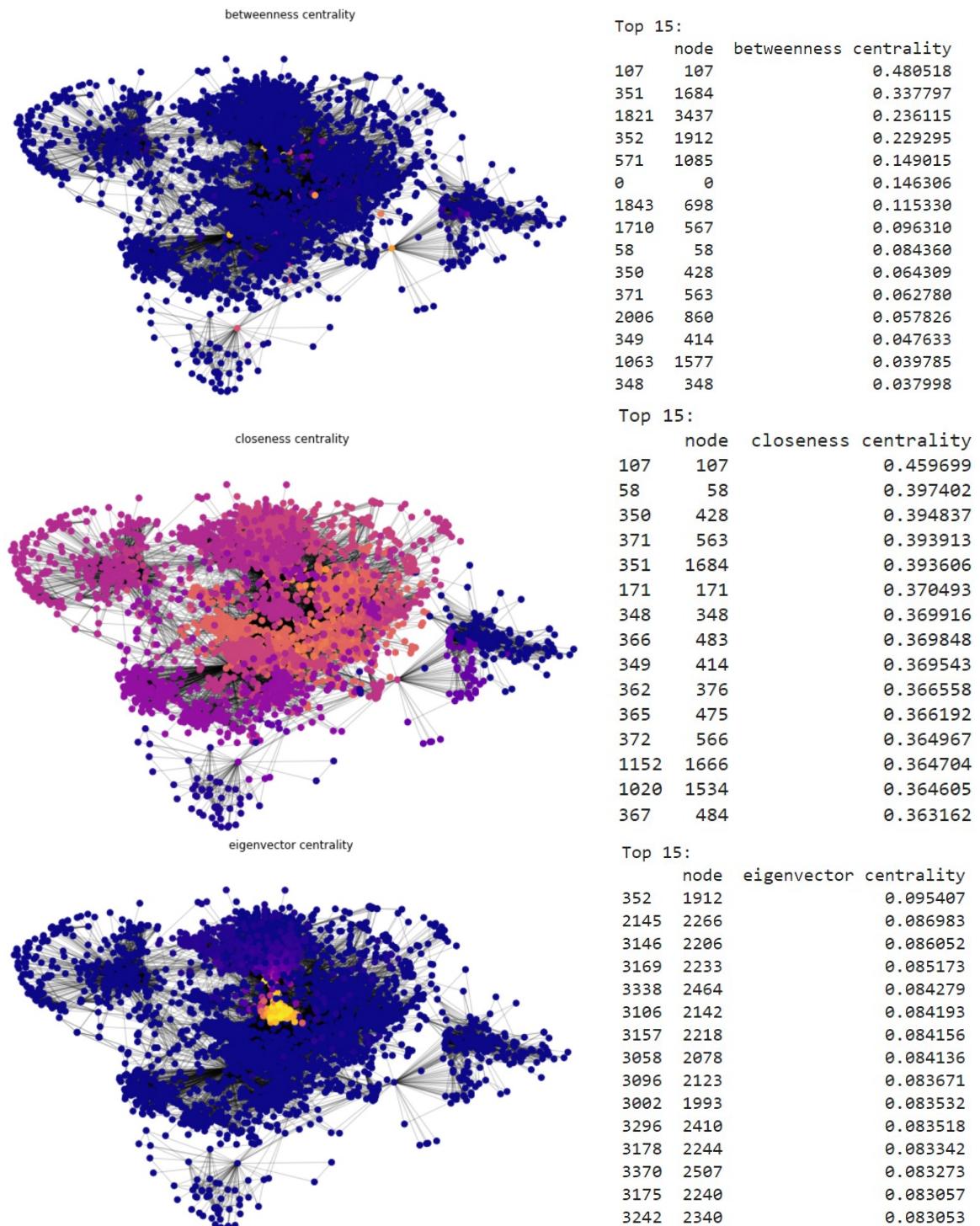
In una rete modestamente grande, un aumento del **t_sus**, i nodi torneranno più lentamente nello stato di "Susceptile", dunque i nodi infetti avranno più difficoltà a trasmettere la malattia.

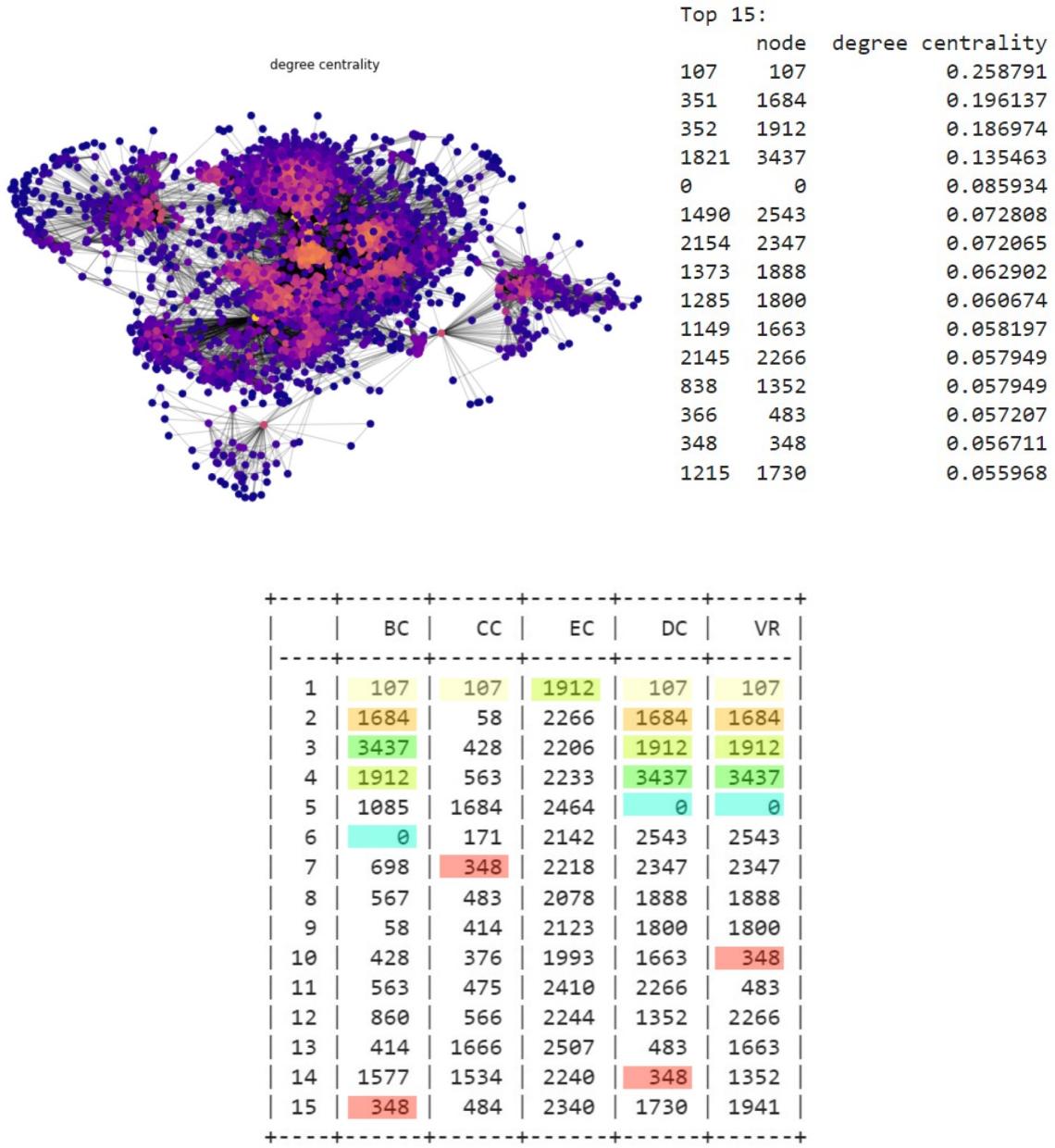
Con una **p_trans** elevata si ha naturalmente un maggiore tendenza e velocità a trasmettere la malattia nei singoli timestep, ciò non togliendo l'importanza dei nodi influenti e più influenti che possono ancora di più contribuire alla velocità dell'infezione dei nodi. In questa simulazione, si nota il caso estremo: i nodi vengono tutti infettati, la malattia

non può più essere espansa; passati i `t_rec` e `t_sus`, i nodi rimangono per sempre nello stato "Susceptible".

4.1.1 Facebook super-spreaders

Per l'analisi dei nodi super-spreader si procede utilizzando i metodi `heatmap_centrality()` e `super_spreaders()`. Dato che queste funzioni lavorano direttamente con la struttura effettiva della rete, si possono richiamare su qualsiasi modello simulativo creato.





Dalle heatmap di centralità dei nodi si possono fare alcune osservazioni:

- la rete ha un buon valore di **betweeness centrality** ma non con un elevato numero di nodi, ciò indica che non ci sono molti nodi influenti nelle le shortest path del network
- la rete ha buoni valori di **closeness centrality** per i nodi, si hanno molti collegamenti interni con la presenza di buone shortest-path
- I valori di **eigenvector centrality** mostrano la presenza di una piccola ristretta cerchia di nodi di discreta importanza, che però non hanno molta influenza nella completa estensione della rete
- sono presenti buoni valori di **degree centrality**, dunque una buona parte di nodi hanno un buon numero di nodi vicini

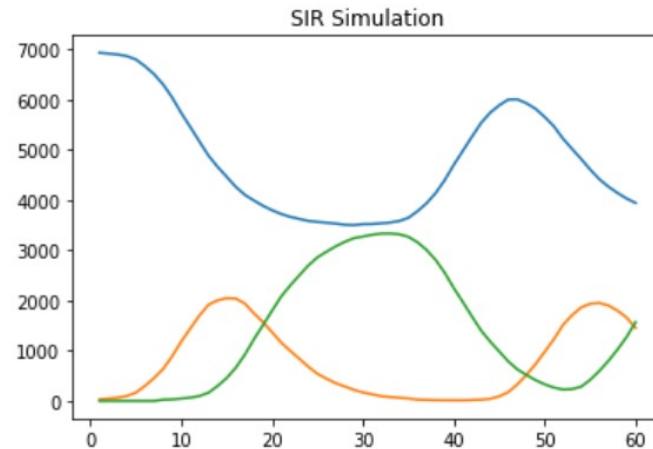
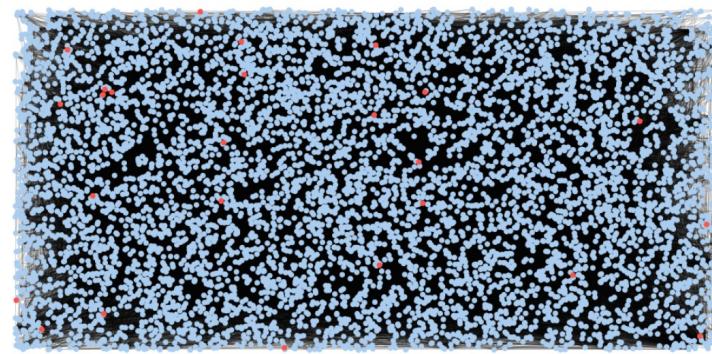
In questa prima rete analizzata, la facebook network, si ha una dimensione molto minore, dunque si tratta di una big network basilare: qui ne caratterizzano la presenza di numerosi nodi con elevato numero di vicini e il basso valore di nodi influenti nelle shortest-path; in virtù di questo, avremo più variazione su quali siano i nodi influenti per ciascuna metrica analizzata, e quindi in tale rete sono presenti si alcuni nodi super-spreader, ma esiste una maggioranza di nodi con "minor influenza" che riescono a comunicare in tutto il diametro della rete

4.2 Twitch network

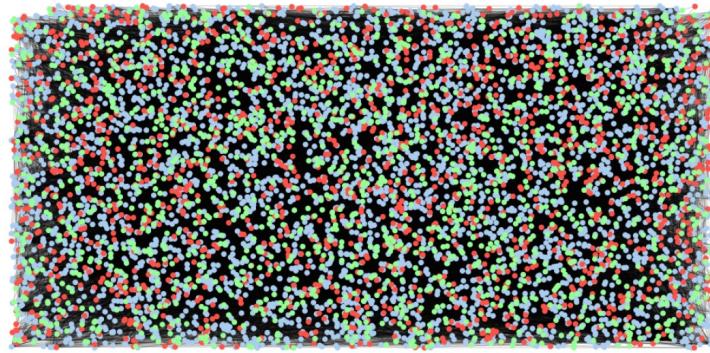
La rete in analisi è maggiormente più grande con 6960 nodi, dunque possono delinearsi risultati diversi.

Risultati di alcune simulazioni:

```
#Simulazione con probabilità di infezione al 2%
twitch1 = Sir(25, 'musae_ENGB.edges.csv', delimiter=',')
twitch1.simulation(0.02, 7, 21, 60) #input parametri simulazione:
    p_trans, t_rec, t_sus, t_sim
```

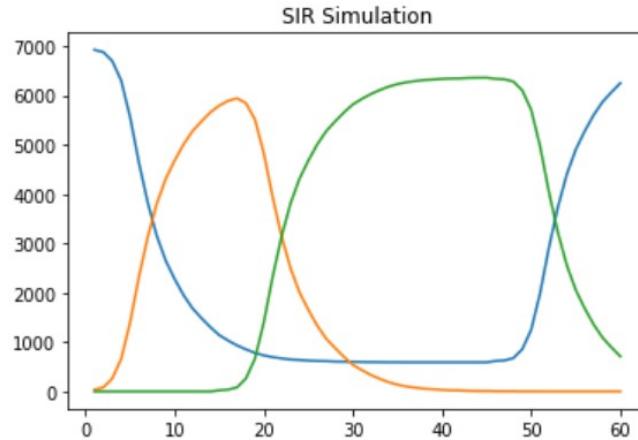
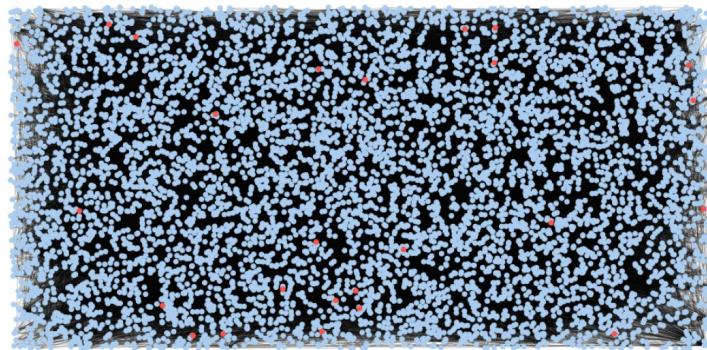


```
Giorno 60 :
Suscettibili: 3940
Infetti: 1452
Guariti: 1568
```

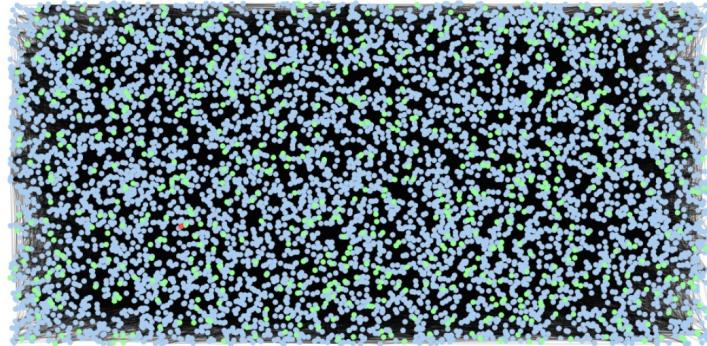


Si può notare che in questa network più grande, impostando questa infezione, si riescono ad infettare pochi nodi. Anche raddoppiando il numero di infetti iniziali il risultato è pressochè simile. Ciò potrebbe indicare che nella rete ci sono pochi nodi influenti oppure che la rete, nel complesso, ha un bassa velocità di trasmissione (scarso livello di shortest-path).

```
#Simulazione probabilità trasmissione più alta: 7%
twitch2 = Sir(25, 'musae_ENGB.edges.csv', delimiter=',')
twitch2.simulation(0.07, 7, 21, 60) #input parametri simulazione:
    p_trans, t_rec, t_sus, t_sim
```



Giorno 60 :
 Suscettibili: 6247
 Infetti: 1
 Guariti: 712

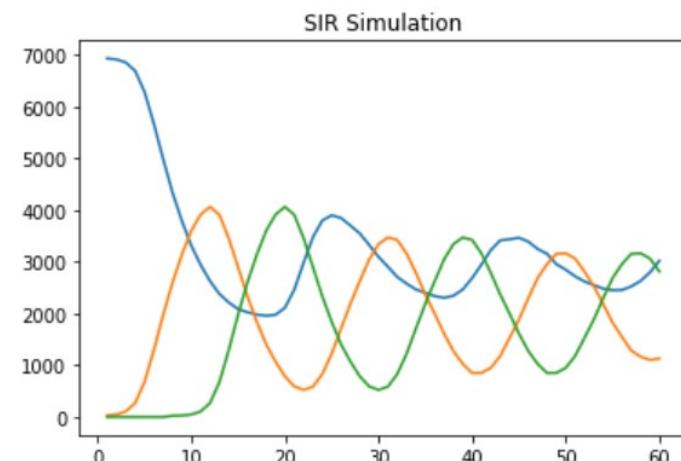
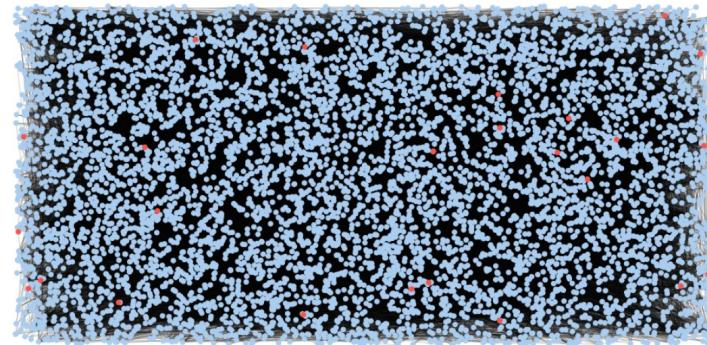


Aumentando il valore di `p_trans` si nota immediatamente come la trasmissione della malattia ha un impatto maggiore.

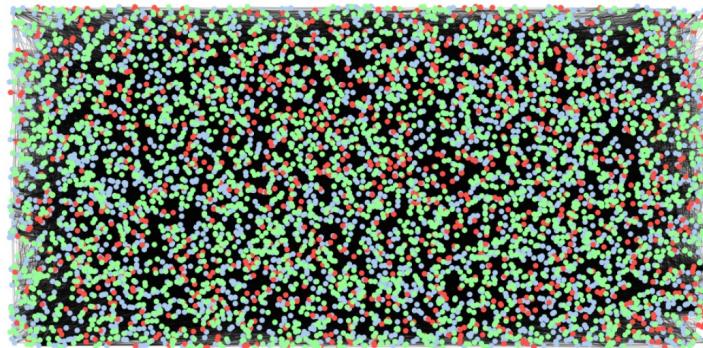
```
#Simulazione con t_rec e t_sus bassi.

twitch3 = Sir(25, 'musae_ENGB_edges.csv', delimiter=',')
twitch3.simulation(0.05, 7, 7, 60) #input parametri simulazione:p_trans
, t_rec, t_sus, t_sim
```

La malattia, con questi parametri, dovrebbe continuare sempre ad espandersi: `t_rec` e `t_sus` bassi, permettono ad un nodo di tornare velocemente allo stato "Susceptible" e quindi con più probabilità di ritornare infetto.

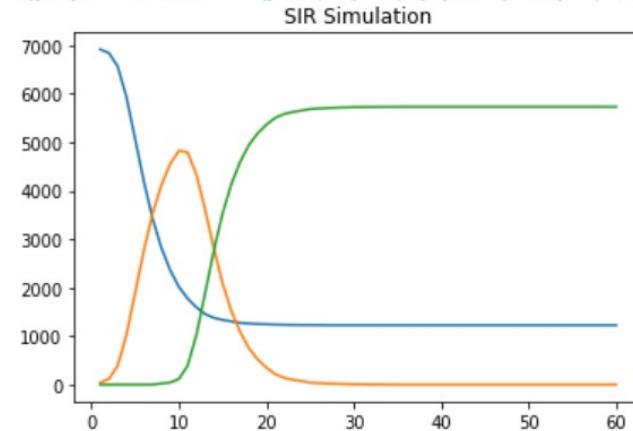
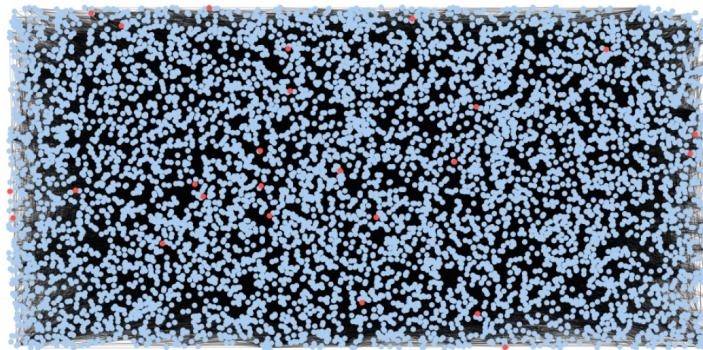


```
Giorno 60 :
Suscettibili: 3019
Infetti: 1128
Guariti: 2813
```

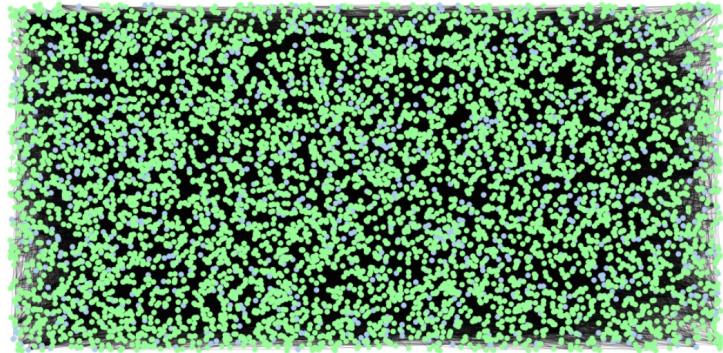


```
#Simulazione con t_sus -> infinito (999)

twitch4 = Sir(25, 'musae_ENGB_edges.csv', delimiter=',')
twitch4.simulation(0.07, 7, 999, 60) #input parametri simulazione:
    p_trans, t_rec, t_sus, t_sim
```



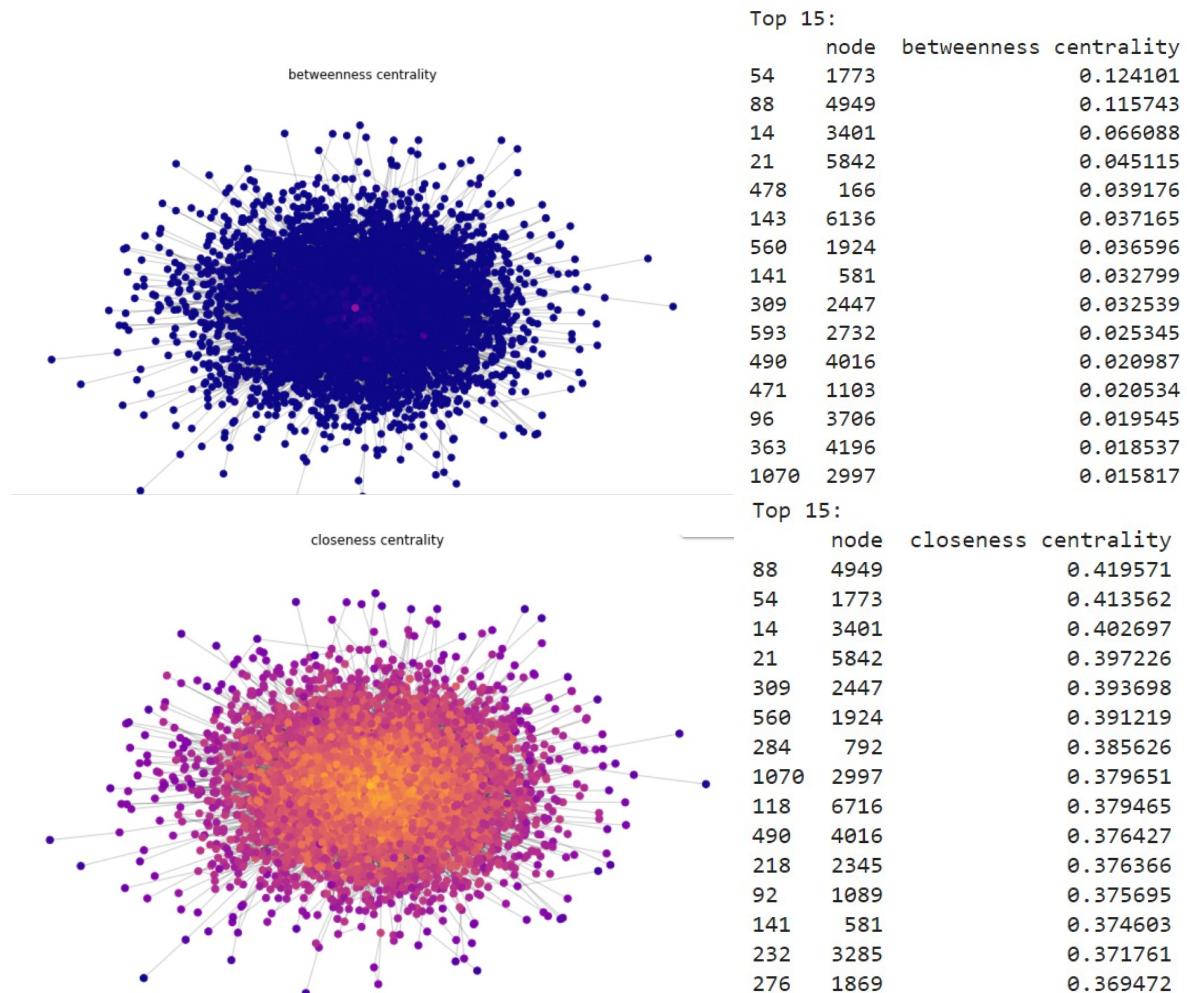
Giorno 60 :
Suscettibili: 1228
Infetti: 0
Guariti: 5732

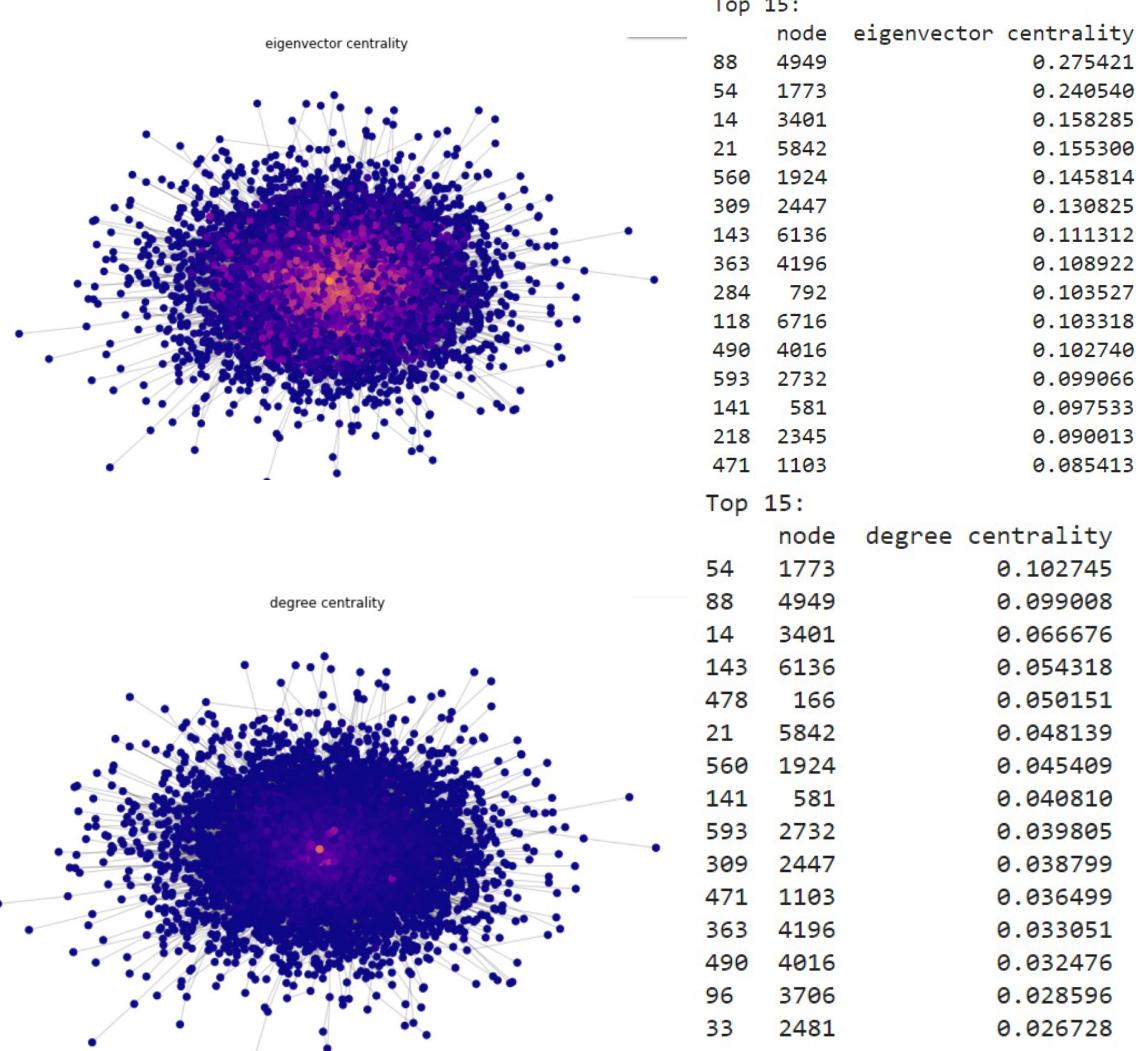


Con questo caso particolare, avremo una guarigione completa con immunità. I nodi guariti non saranno più vulnerabili alla malattia; i nodi infetti di conseguenza, non potendo trasmettere la malattia, tenderanno alla scomparsa.

4.2.1 Twitch super-spreaders

Anche per questa rete, per evidenziare inodi super-spreader si procede utilizzando i metodi `heatmap_centrality()` e `super_spreaders()`. Dato che queste funzioni lavorano direttamente con la struttura effettiva della rete, si possono richiamare su qualsiasi modello simulativo creato.





	BC	CC	EC	DC	VR
1	1773	4949	4949	1773	1773
2	4949	1773	1773	4949	4949
3	3401	3401	3401	3401	3401
4	5842	5842	5842	6136	6136
5	166	2447	1924	166	166
6	6136	1924	2447	5842	5842
7	1924	792	6136	1924	1924
8	581	2997	4196	581	581
9	2447	6716	792	2732	2732
10	2732	4016	6716	2447	2447
11	4016	2345	4016	1103	1103
12	1103	1089	2732	4196	4196
13	3706	581	581	4016	4016
14	4196	3285	2345	3706	3706
15	2997	1869	1103	2481	2481

Dalle heatmap di centralità dei nodi di possono fare alcune osservazioni:

- la rete ha un basso valore di nodi con **betweeness centrality**, ciò indica che ci sono pochissimi nodi influenti in tutte le shortest path
- la rete ha elevatissimi valori di **closeness centrality** per i nodi, dunque la rete è ben collegata da moltissimi nodi con ottime shortest-path;
- ci sono buoni valori di **eigenvector centrality**, dunque sono presenti abbastanza nodi importanti vicini ad altri nodi importanti soprattutto nella parte centrale della rete.
- i valori di **degree centrality** sono molti bassi, indica che nella rete i nodi hanno pochissimi vicini

In questa rete dunque, abbiamo un importante gruppo di nodi super-spreader che si concentrano nella maggiore parte centrale, mentre nelle sottoreti più limitrofe si può notare che i livelli di influenza sono sempre minori. Ciò nonostante, nella rete abbiamo una costanza su quali nodi sono ritenuti più importanti e influenti per la trasmissione attraverso le path.

Un importante analisi è il rapporto tra i valori di **closeness centrality** e **degree centrality**: nonostante i valori di degree centrality siano bassi, quindi indicando che i nodi hanno pochi vicini, abbiamo importantissimi valori di closeness. Questa relazione indica che nonostante i nodi abbiano pochi vicini, il loro legame è molto alto dove moltissimi nodi hanno nodi vicini comuni.