

Capitolo 7

Analisi del modello discreto

7.1 Modello [4] di S. Belen e C.E.M. Pearce

7.1.1 Due linee di pensiero

Nell'articolo [4] viene presentata una versione discreta del modello basata su un sistema di transizioni che regola i passaggi di stato. Gli autori fanno riferimento a ciò che hanno trovato in letteratura. Una certa popolazione può essere associata ad un grafo, dove i vertici rappresentano gli individui mentre gli archi del grafo rappresentano le amicizie/collegamenti. Il grafo considerato $G = (V, E)$ ha $N := |V| \in \mathbb{N}$ vertici, inoltre come visto in precedenza supponiamo di partizionare la popolazione in tre sottoinsiemi I -Ignorants, S -Spreader e R -Stiflers avendo $I, S, R \subseteq V$. Gli stati del sistema sono triple del tipo (i, s, r) . La terna $(i, s, r) \in \mathbb{N}^3$ rappresenta la situazione/stato delle sottopopolazioni durante le transizioni dell'automa, la somma $i + s + r = |V|$ è la cardinalità della popolazione totale. $i := |I|$ rappresenta la cardinalità della sottopopolazione di Ignorants, $s := |S|$ è la cardinalità della sottopopolazione di Spreaders e $r := |R|$ è la cardinalità della sottopopolazione di Stiflers.

Per quanto riguarda lo stato iniziale, siamo stati costretti a ipotizzare varie ipotesi in quanto sull'articolo originale non è chiaro:

i) Esiste uno (e un solo) Spreader iniziale: questa ipotesi è conforme alla situazione, molto diffusa, in cui una notizia è generata da un'unica fonte. Tale ipotesi viene senz'altro assunta.

ii)(a) Secondo l'articolo i soggetti sono classificabili, prima che entrino in interazione, come Stifler o Ignorants. Questa ipotesi modella una situazione in cui, indipendentemente dal carattere della notizia, un individuo reagisce in qualità di Ignorant o Stifler. Questo viene determinato a priori, quindi prima

che tutto abbia inizio.

ii)(b) I soggetti sono classificabili come Stiflers o Ignorants solo dopo essere entrati in interazione. Questa ipotesi modella la situazione in cui i soggetti, prima ascoltano la notizia, e poi valutano se diventare Stiflers oppure Spreaders.

Nella tesi abbiamo preso in considerazione entrambi gli scenari *ii)(a)* e *ii)(b)*. Una volta fissate le condizioni iniziali, il comportamento del sistema è determinato dalle seguenti transizioni: riassumendo si propone l'algoritmo del modello [4] fedele a come si trova in letteratura:

Spreader interagisce con un Ignorant: l'Ignorant è portato a diventare Spreader $(i, s, r) \rightarrow (i - 1, s + 1, r)$

Spreader interagisce con uno Stifler: lo Spreader si converte in Stifler $(i, s, r) \rightarrow (i, s - 1, r + 1)$

Spreader interagisce con uno Spreader: accade che si annullano a vicenda diventando entrambi Stiflers $(i, s, r) \rightarrow (i, s - 2, r + 2)$

7.2 Alcuni Risultati

Sono state fatte simulazioni per entrambi programmi *ii)(a)* Stiflers non preesistenti e *ii)(b)* Stiflers pre esistenti. Nei grafici abbiamo riportato le medie delle simulazioni degli stati finali $(i, 0, r)$. Abbiamo fatto questo lasciando fisso il numero dei vertici a 700 nodi e 1200 archi per i Grafi Random. Queste simulazioni sono state con la probabilità del 50% per gli Ignorants di convertirsi in Stiflers.

Tutte le simulazioni considerano i nodi e gli archi di un certo grafo $G = (V, E)$ dove $|V| = N = 700$ nodi per il grafo e $|E| = 12000$ archi, il grafo completo ha invece $\frac{N(N-1)}{2} = (700^2 - 700)\frac{1}{2} = 244650$ archi. La scelta dei 700 nodi e 12000 archi ci è sembrato un buon compromesso per non appesantire troppo il carico di lavoro della CPU e allo stesso tempo avere un grafo che avesse un numero ragionevole di nodi connessi tra loro per poter operare una simulazione. Con la lettera K si intendono gli Spreaders Cumulativi, mentre con C si intendono le azioni totali compiute dagli Spreaders. Con SpreadRate si intende il rapporto tra Spreaders Cumulativi e popolazione totale. Vediamo quindi di seguito alcuni risultati della versione *ii)(a)* del modello [4] Stiflers quisenzandi pre esistenti.

| GRAFO COMPLETO STATICO | | | | | | | Spread-Rate |
|------------------------|---|-----|-----|-----|-----|--------|---------------------|
| i | s | r | C | K | V | E | |
| 329 | 0 | 371 | 530 | 180 | 700 | 244650 | 0.2571428571428571 |
| 335 | 0 | 365 | 519 | 176 | 700 | 244650 | 0.25142857142857145 |
| 290 | 0 | 410 | 600 | 228 | 700 | 244650 | 0.32571428571428573 |

GRAFO COMPLETO (Con matrice di adiacenza con variazione del 50%)

| i | s | r | C | K | V | E | Spread-Rate |
|--|---|-----|-----|-----|-----|--------|---------------------|
| 358 | 0 | 342 | 480 | 162 | 700 | 244650 | 0.23142857142857143 |
| 342 | 0 | 358 | 508 | 173 | 700 | 244650 | 0.24714285714285714 |
| 327 | 0 | 373 | 548 | 207 | 700 | 244650 | 0.2957142857142857 |
| GRAFO INCOMPLETO STATICO | | | | | | | |
| i | s | r | C | K | V | E | Spread-Rate |
| 343 | 0 | 357 | 511 | 185 | 700 | 12000 | 0.2642857142857143 |
| 474 | 0 | 226 | 317 | 114 | 700 | 12000 | 0.16285714285714287 |
| 367 | 0 | 333 | 476 | 173 | 700 | 12000 | 0.24714285714285714 |
| GRAFO INCOMPLETO (Con matrice di adiacenza con variazione del 50%) | | | | | | | |
| i | s | r | C | K | V | E | Spread-Rate |
| 320 | 0 | 380 | 544 | 197 | 700 | 12000 | 0.2814285714285714 |
| 341 | 0 | 359 | 511 | 179 | 700 | 12000 | 0.2557142857142857 |
| 326 | 0 | 374 | 540 | 193 | 700 | 12000 | 0.2757142857142857 |

Di seguito i risultati della simulazione della versione *ii)(b)* del modello [4] con Stiflers pre esistenti.

| GRAFO COMPLETO STATICO | | | | | | | |
|--|---|-----|----|----|-----|--------|-----------------------|
| In questo caso abbiamo un grafo completo, ma resta invariato per tutta la simulazione. | | | | | | | |
| i | s | r | C | K | V | E | Spread-Rate |
| 362 | 0 | 338 | 1 | 1 | 700 | 244650 | 0.0014285714285714286 |
| 308 | 0 | 392 | 68 | 35 | 700 | 244650 | 0.05 |
| 368 | 0 | 332 | 1 | 1 | 700 | 244650 | 0.0014285714285714286 |
| GRAFO COMPLETO (Con matrice di adiacenza con variazione del 50%) | | | | | | | |
| In questo caso abbiamo un grafo completo, ma la sua matrice di adiacenza varia ad ogni iterazione del 50%. | | | | | | | |
| i | s | r | C | K | V | E | Spread-Rate |
| 345 | 0 | 355 | 1 | 1 | 700 | 244650 | 0.0014285714285714286 |
| 361 | 0 | 339 | 5 | 3 | 700 | 244650 | 0.004285714285714286 |
| 331 | 0 | 369 | 1 | 1 | 700 | 244650 | 0.0014285714285714286 |
| GRAFO INCOMPLETO STATICO | | | | | | | |
| In questa situazione il grafo G viene generato in modo aleatorio, ma resta invariato per tutta la simulazione. | | | | | | | |
| i | s | r | C | K | V | E | Spread-Rate |
| 354 | 0 | 346 | 17 | 9 | 700 | 12000 | 0.012857142857142857 |
| 295 | 0 | 405 | 78 | 41 | 700 | 12000 | 0.05857142857142857 |
| 335 | 0 | 365 | 1 | 1 | 700 | 12000 | 0.0014285714285714286 |
| GRAFO INCOMPLETO (Con matrice di adiacenza con variazione del 50%) | | | | | | | |
| In questa situazione il grafo G viene generato in modo aleatorio, ma la sua matrice di adiacenza varia ad ogni iterazione del 50%. | | | | | | | |
| i | s | r | C | K | V | E | Spread-Rate |
| 336 | 0 | 364 | 11 | 6 | 700 | 12000 | 0.008571428571428572 |
| 340 | 0 | 360 | 6 | 4 | 700 | 12000 | 0.005714285714285714 |
| 327 | 0 | 373 | 3 | 2 | 700 | 12000 | 0.002857142857142857 |

Dai risultati ottenuti mediante le simulazioni al computer è emerso che in media ogni Spreader è in grado di diffondere il rumor in media per 2.5 volte prima della sua estinzione nel caso *ii)(a)*. Invece in media ogni Spreader è in grado di diffondere il rumor in media per 1.5 volte prima di estinguersi nella versione *ii)(b)*.

Con il professore Shuichi Miyazaki della Kyoto University abbiamo in par-

icolare studiato il modello $ii)(a)$ in quanto non ci sono Stiflers pre esistenti. Sono stati studiati quattro casi: Grafo Random Statico, Grafo Random con matrice di adiacenza variabile, Grafo Completo Statico e Grafo Completo con matrice di adiacenza variabile. Sul suggerimento del professore Miyazaki abbiamo fatto variare il parametro p , dove p è la probabilità con cui un Ignorant contattato da uno Spreader si converte in Spreader, di conseguenza $1 - p$ è la probabilità per l'Ignorant di convertirsi in Stifler. Rispetto a quanto prima fatto, in queste simulazione i Grafi Random hanno un numero random anche di archi, presi nell'intervallo di numeri naturali $\{2|V|, \dots, \frac{|V|^2 - |V|}{2}\} \subset \mathbb{N}$, dato che il generico grafo $G = (V, E)$ ha $|V|$ nodi ed $|E|$ archi. Abbiamo fatto centinaia di simulazioni facendo variare la probabilità portando quindi p ad assumere i valori 1%, 10%, 20%, 30%, 50%, 70%, 90% e 99%. Sono stati mantenuti in tutte le simulazioni il numero di nodi costante a 700, quindi il range degli archi risulta essere $|E| \in \{1400, \dots, 244650\} \subset \mathbb{N}$. Negli stati finali delle simulazioni ci troviamo ad avere quindi $i + r = 700$, per ogni probabilità considerata abbiamo fatto la media degli stati finali della forma $(i, 0, r)$. Dalle simulazioni è risultato che non vi è molta differenza rispettivamente tra Grafo Random e Grafo Random con matrice di adiacenza variabile, Grafo Completo e Grafo Completo con matrice di adiacenza variabile. A questo fatto non siamo riusciti a dare una valida motivazione, ci siamo limitati riportare solamente i risultati. Abbiamo lasciato invariato il fatto che nei casi di variazione della matrice di adiacenza questa faccia il complementare sul 50% (discretizzato) delle coppie di nodi. Dalle simulazioni al variare di p invece è emerso che all'aumentare di p aumentano anche gli Stiflers finali, mentre al diminuire di p diminuiscono anche gli Stiflers finali. Inoltre abbiamo osservato che gli Stiflers finali sono inferiori agli Ignorants finali. Avviene un cross tra il grafico delle cardinalità dei nodi finali di Ignorants e il grafico delle cardinalità dei nodi finali di Stiflers al crescere di p . Quindi risulta che gli Stiflers finali superano gli Ignorants finali con p intorno al 50% in tutti i quattro i casi. Anche se il cross avviene un po' più lentamente con il Grafo Completo sia che abbia la matrice di adiacenza variabile o statica. A questi fatti non abbiamo saputo dare una motivazione. Questi risultati ci sono sembrati ragionevoli come si evince dalle figure 7.1, 7.2, 7.3 e 7.4. Ci sono sembrati risultati ragionevoli anche il fatto, come si vede dalle tabelle, che all'aumentare degli Spreaders cumulativi ogni Spreader singolarmente in media diffonde poche volte il rumor. Mentre al diminuire degli Spreaders cumulativi ogni Spreader singolarmente in media diffonde tante volte il rumor. Anche questi risultati ci sono sembrati del tutto ragionevoli come si osserva dalle figure 7.5 e 7.6.

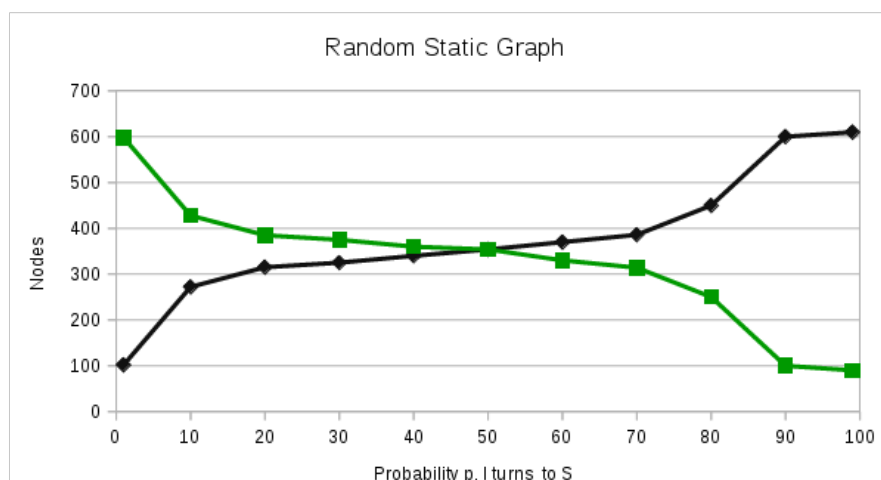


Figura 7.1: Tabella per il Grafo Random Statico, la linea VERDE rappresenta gli Ignorants mentre la linea in NERO rappresenta gli Stiflers.

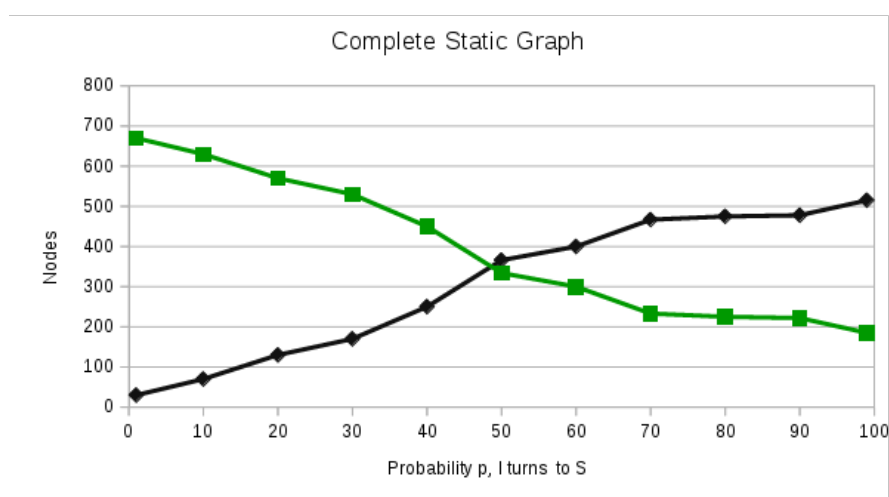


Figura 7.2: Tabella per il Grafo Completo Statico, la linea VERDE rappresenta gli Ignorants mentre la linea in NERO rappresenta gli Stiflers.

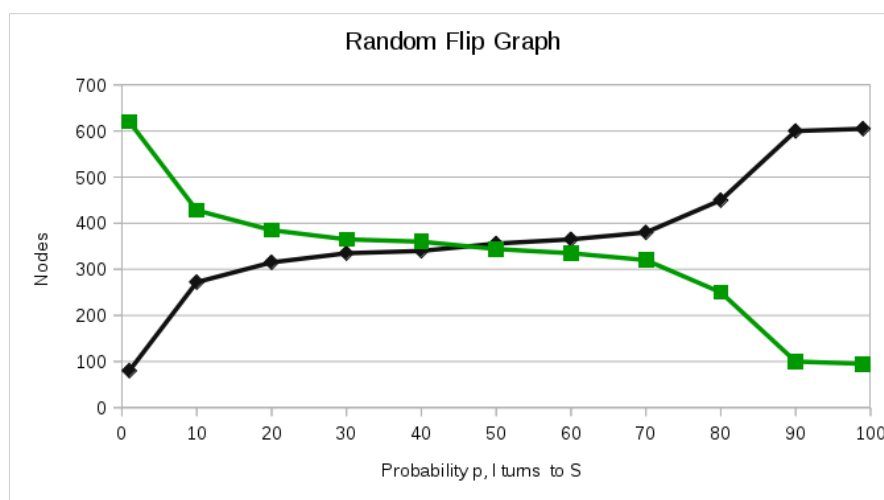


Figura 7.3: Tabella per il Grafo Random Variabile, la linea VERDE rappresenta gli Ignorants mentre la linea in NERO rappresenta gli Stiflers.

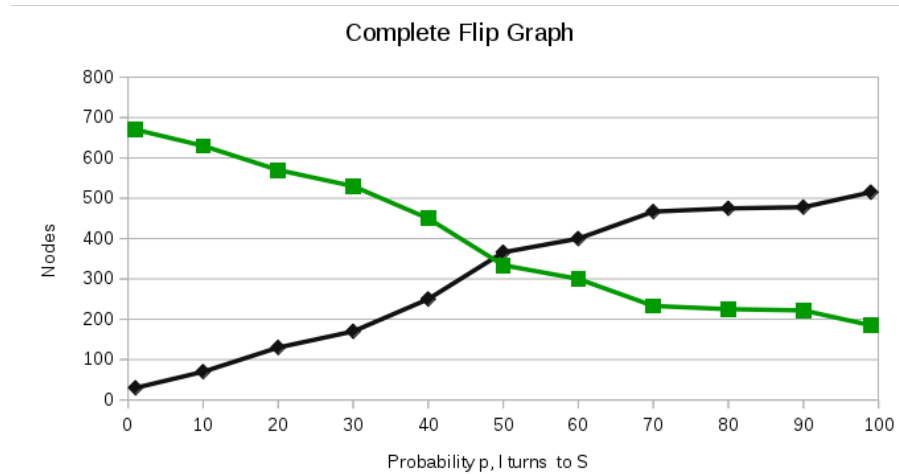


Figura 7.4: Tabella per il Grafo Completo Variabile, la linea VERDE rappresenta gli Ignorants mentre la linea in NERO rappresenta gli Stiflers.

| p | Average Rounds | Total | Average Total Spreaders | Average Rounds for single Spreader |
|-----|----------------|-------|-------------------------|------------------------------------|
| 1% | 55.3 | | 1.5 | 36.8 |
| 10% | 102.7 | | 1.5 | 68.4 |
| 30% | 350.4 | | 98.6 | 3.55 |
| 50% | 551.3 | | 197.4 | 2.79 |
| 70% | 732.7 | | 341.9 | 2.14 |
| 90% | 820.8 | | 439.6 | 1.86 |
| 99% | 901.6 | | 468.8 | 1.9 |

Figura 7.5: Tabella per il Grafo Completo Statico.

| p | Average Rounds | Total | Average Total Spreaders | Average Rounds for single Spreader |
|-----|----------------|-------|-------------------------|------------------------------------|
| 1% | 29.7 | | 1.5 | 19.8 |
| 10% | 58.5 | | 9.8 | 5.9 |
| 30% | 310.2 | | 68.7 | 4.51 |
| 50% | 490.4 | | 116.7 | 4.2 |
| 70% | 701.2 | | 291.3 | 2.4 |
| 90% | 894.8 | | 505.8 | 1.76 |
| 99% | 903.6 | | 508.9 | 1.77 |

Figura 7.6: Tabella per il Grafo Random Statico.

Capitolo 8

Allegato codice

8.1 Codice per il modello [4]

8.1.1 Descrizione

Per implementare l'automa abbiamo pensato di utilizzare come struttura dati, per rappresentare il grafo associato (dove i nodi rappresentano gli individui della popolazione e gli archi rappresentano i collegamenti telematici fra i soggetti), una matrice di adiacenza. Dato quindi il numero totale di nodi, $N \in \mathbb{N}$ è un numero naturale rappresentante la cardinalità della popolazione sottesa. Abbiamo quindi creato una matrice simmetrica a coefficienti 0, 1 con diagonale nulla di dimensione $N \times N$. Generalmente creata da un algoritmo random, ad eccezione per il caso del grafo completo dove la matrice di adiacenza ha tutti gli elementi unitari con diagonale nulla. Per generare le matrici di adiacenza abbiamo creato i metodi *CreaGrafoRandom()* e *CreaGrafoCompleto()*. Inoltre occorre anche tenere in memoria i nodi e i loro relativi attributi: *id*-numero identificativo, colore, x e y coordinate per la rappresentazione grafica. Per poter memorizzare gli attributi di ogni nodo abbiamo creato una matrice $N \times 3$ che viene chiamata *RegistroColoriCoord*. Essa tiene conto di tutti gli attributi di tutti i nodi. Nel momento della creazione di *RegistroColoriCoord* viene assegnato ad ogni nodo un *id* assegnato dal numero della riga, le righe vanno da 0 a $N - 1$, abbiamo che $\forall j = 0, \dots, N - 1$ il numero identificativo id_j associato al j -esimo nodo vale $id_j = j$. Nella prima colonna viene inserito il colore che può essere: bianco-Stifler, verde-Ignorant e rosso-Spreader. Tuttavia ai colori abbiamo associato tre numeri naturali *bianco* = 0, *verde* = 1 e *rosso* = 2, dato che nel programma lavoriamo con numeri e non con stringhe. Vengono successivamente assegnate le coordinate cilindriche, per il “rendering” grafico, per stampare i nodi a schermo sparsi a spirale. Le coordinate sfruttano il numero della riga (corrispondente all'*id* del nodo), quindi le coordinate

cilindriche assegnate al j -esimo nodo con $id_j = j$ sono: $x_j = r_j \cos(j) + \frac{1}{2}$ e $y_j = r_j \sin(j) + \frac{1}{2}$. Il valore $r_j := \frac{j}{2N}$ è un fattore per portare in scala la stampa dei nodi a schermo, mentre lo scostamento $\frac{1}{2}$ occorre in quanto la libreria grafica *pygame* utilizzata prevede l'origine nell'angolo in alto a sinistra e quindi questo serve per portare il nodo con $id = 0$ al centro della spirale di Archimede. Utilizzando l'id dei vertici $\forall j = 0, \dots, |V| - 1$, per generare le posizioni, risulta che le coppie delle coordinate nel piano cartesiano sono $(x_j * 1024, y_j * 800)$. Dove $1024 * 800$ risulta essere la dimensione dello schermo per portare la spirale di Archimede in proporzione alla risoluzione del display. Quando i nodi sono stampati sullo schermo vengono rappresentati come dei piccoli dischi colorati. Quando i dischi vengono stampati a schermo le coordinate delle loro posizioni vengono espanse moltiplicando rispettivamente x e y con dei fattori di scala basati sulla dimensione della finestra che incorpora la grafica, $x \cdot 1024$ e $y \cdot 800$. Nel caso *ii*)(a) (con riferimento al Capitolo 5, sottosezione 5.1.1) sono stati creati i metodi: *SmeetI*()-Spreader interagisce con Ignorant associato alla transizione $(i, s, r) \rightarrow (i - 1, s + 1, r)$, *SmeetS*()-Spreader interagisce con Spreader associato alla transizione $(i, s, r) \rightarrow (i, s - 2, r + 2)$ e *SmeetR*()- Spreader interagisce con Stifler associato alla transizione $(i, s, r) \rightarrow (i, s - 1, r + 1)$. Per creare la situazione iniziale c'è il metodo *RegistroColoriCoordVerdi*() per colorare tutti i nodi di verde. Mentre *CreaScenario*() assegna casualmente degli Stifler (il 50% dei nodi circa) e uno Spreader di partenza. Il cuore del programma è il metodo *ComputingRandomNoGUI*(), senza un output grafico per risparmiare cicli della CPU nel calcolo, seleziona dal sottoinsieme degli Spreaders uno Spreader a caso e un nodo a caso tra i contatti di quest'ultimo. Dopo aver memorizzato la coppia di nodi scelta, si pone un condizionale: se il nodo in contatto con lo Spreader è un'Ignorant viene invocata *SmeetI*(), se è uno Spreader viene invocata *SmeetS*() e se si tratta di uno Stifler viene invocata *SmeetR*(). Il metodo *ComputingRandomNoGUI*() viene invocato dai metodi che stampano i files *csv* fin quando ci sono Spreaders nel *RegistroColoriCoord*. quindi il programma termina quando gli Spreaders finiscono. Da parte nostra abbiamo aggiunto in taluni casi la possibilità che la matrice di adiacenza possa variare con una certa percentuale scelta, il metodo invocato per compiere ciò è *CambiaMatrice*() il quale selezionando una coppia di nodi a caso, crea il collegamento qualora non ci sia e rimuove il collegamento qualora ci sia. La percentuale di archi che vogliamo cambiare viene stimata iterando *CambiaMatrice*() per un numero di volte pari alla percentuale scelta discretizzata $\lfloor N \cdot percentuale \rfloor$.

Il caso *ii*)(b) è abbastanza simile al precedente come script, tuttavia sono state apportate alcune modifiche, il metodo *SmeetI*() si suddivide in *SmeetINC*()-Spreader interagisce con Ignorant Non Contagiabile $(i, s, r) \rightarrow (i - 1, s, r + 1)$,

la differenza consiste che questa volta lo Spreader non si estingue subito dopo la colorazione del nodo con cui ha interagito. Così *SmeetIC()*-Spreader interagisce con Ignorant Contagiabile $(i, s, r) \rightarrow (i-1, s+1, r)$, dopo la colorazione anche questa volta lo Spreader non si estingue. Lo Spreader subisce mutazioni solo nel caso *SmeetS()* o *SmeetR()* che restano invariate come prima. Il metodo *ComputingRandomNoGUI()* resta semanticamente inalterato, solo nello script ci sono alcuni ritocchi sugli operatori condizionali. In quest'ultimo caso il metodo *CreaScenario()* si limita a posizionare uno Spreader iniziale.

8.1.2 Codice sorgente

Viene riportato il sorgente in Python 2.7.8 col quale sono state fatte le simulazioni. Le istruzioni sono scritte al fine di far funzionare l'automa quindi non sono state ottimizzate. Inoltre sono state sfruttate librerie già presistenti per poter alleggerire il carico del lavoro informatico. Come ad esempio: la libreria *numpy* per la gestione di vettori e matrici, la libreria *csv* per poter esportare i dati elaborati in formato foglio elettronico e la libreria *pygame* per la parte grafica. Pertanto laddove fosse stato possibile trovare componenti aggiuntive utili al fine di realizzare il programma non abbiamo “reinventato la ruota”.

Sorgente associato allo scenario *ii)a)* senza Stiflers pre esistenti:

```
# -*- coding: utf-8 -*- #

import math
import numpy
import random
import pygame
import csv

##### VALORI MODIFICABILI DALL'UTENTE #####
# Sia G = (V,E) grafo
nodes = 700 # |V| = nodes (vertici)
ite = 5 # numero di iterazioni
percentuale = 0.5 # percentuale degli archi modificati
#####

edges = random.sample(range(2*nodes, int((nodes*nodes-nodes)*0.5)), 1)[0] # |E| = edges (archi)
valore = int(edges*percentuale) # calcolo percentuale discretizzata
S = 0 # inizializzazione degli Spreaders
R = 0 # inizializzazione Stiflers
K = 1 # Spreaders cumulativi
C = 0 # cicli totali azioni compiute dagli Spreaders
SpreaderRate = K / nodes
rosso = 2 # il rosso nella grafica è associato agli Spreaders
verde = 1 # il verde nella grafica è associato agli Ignorants
bianco = 0 # il bianco nella grafica è associato agli Stiflers
RegistroColoriCoord = None

def SmeetIC(): # (I,S,R) |- (I-1, S+1,R)

    global I
    global S
    global C
    global K
    I=I-1
    S=S+1
    C=C+1 # Segno in C che è avvenuta una azione di SPREAD
    K=K+1 # Solo in SmeetIC() gli Spreaders aumentano, quindi lo segno nel contatore K
    print "S vs I susceptible",[I,S,R]

def SmeetINC(): # (I,S,R) |- (I-1, S,R+1)
```

```

    global I
    global R
    global C
    global S
    S=S
    I=I-1
    R=R+1
    C=C+1 # Segno in C che è avvenuta una azione di SPREAD
    print "S vs I immune",[I,S,R]

def SmeetS(): # (I,S,R) |- (I, S-2,R+2), transizione Spreader incontra Spreader

    global I
    global S
    global C
    global K
    global R
    I=I
    S=S-2
    R=R+2
    C=C+1
    print "S vs S :",[I,S,R]

def SmeetR(): # (I,S,R) |- (I, S-1,R+1), transizione Spreader incontra Stifler

    global I
    global S
    global C
    global K
    global R
    I=I
    S=S-1
    R=R+1
    C=C+1
    print "S vs R :",[I,S,R]

def RegistroColoriCoordVerdi(): # coloro tutto di verde, quindi resetto lo scenario per
    una nuova iterazione
    global I
    global R
    global S
    global C
    global K
    global nodes
    global RegistroColoriCoord
    RegistroColoriCoord = numpy.zeros(shape=(nodes,3))
    for x in range(0,nodes) :
        i = (x/(2.*nodes))*math.cos(x) + 0.5
        j = (x/(2.*nodes))*math.sin(x) + 0.5
    # le coordinate verranno riproporzionate durante la Colorazione
    (RegistroColoriCoord[x,0])= verde
    (RegistroColoriCoord[x,1])=i
    (RegistroColoriCoord[x,2])=j

    S = 0
    I = nodes
    R = 0
    C = 0
    K = 1

def CreaScenario(): # piazza uno Spreader a random, gli altri restano verdi
    global I
    global S
    global R
    global RegistroColoriCoord
    y = random.sample(range(0,nodes),1)[0]
    RegistroColoriCoord[y,0]= rosso # Scelgo un rosso a caso.

def CreaGrafoCompleto(): # crea una matrice di adiacenza per il grafo completo.
    global M
    global nodes
    M = numpy.zeros(shape=(nodes,nodes))
    for y1 in range(0,nodes):
        for y2 in range(0,nodes):
            M[y1,y2] = M[y2,y1]=1
            M[y1,y1]=0

def CreaGrafoRandom(): # crea una matrice di adiacenza randomizzata
    global M

```

```

global nodes
global edges
x=0
edges = random.sample(range(2*nodes,int((nodes*nodes-nodes)*0.5)),1)[0] # |E| = edges
    (archi)
M = numpy.zeros(shape=(nodes,nodes))
while x<edges :
    i = random.sample(range(0,nodes),1)[0] # associo ad i un valre {0,...,n-1}
        random
    j = random.sample(range(0,nodes),1)[0] # associo ad j un valre {0,...,n-1}
        random
    if (M[i,j] == 1 ) :
        x=x-1
    elif ( M[i,j]== 0 & i!=j ) :
        M[i,j] = M[j,i]=1
    else:
        M[i,i] = 0
        x=x-1
    x=x+1

def ComputingRandomNoGUI() : # Algoritmo casuale
global RegistroColoriCoord
randspreadid=random.sample(numpy.where(RegistroColoriCoord[:,0]==rosso)[0],1)[0]
z = random.sample(numpy.where(M[randspreadid,:]==1)[0],1)[0] # Tra i contatti di
    randspreadid seleziono un nodo a caso connesso

if RegistroColoriCoord[z,0]==verde: # Se il nodo z-esimo selezionato è verde
    if random.uniform(0,1) > 0.99 : # Controllo se è tendenzialmente Rosso
        RegistroColoriCoord[z,0]=rosso # In questo caso coloro il nodo z-esimo di
            Rosso
        SmeetIC()
    else:
        RegistroColoriCoord[z,0]=bianco
        SmeetINC()

elif RegistroColoriCoord[z,0]==bianco :
        RegistroColoriCoord[randspreadid,0]=bianco # Altrimenti se il nodo z-esimo non
            è contagiabile z diventa di colore bianco
        SmeetR()

elif RegistroColoriCoord[z,0]==rosso: # Se il nodo z-esimo è ROSSO
        RegistroColoriCoord[randspreadid,0]=bianco # coloro randspreadid di BIANCO
        RegistroColoriCoord[z,0]=bianco # Coloro il nodo z-esimo di BIANCO
        SmeetS()

def SalvaFileStaticoNormale():
    RegistroColoriCoordVerdi()
    CreaGrafoRandom()
    CreaScenario()

    with open('RandomStat4b.csv', 'w') as csvfile:
        spamwriter = csv.writer(csvfile, delimiter=',', quotechar='|', quoting=csv.
            QUOTE_MINIMAL)
        spamwriter.writerow(['i','s','r','Azioni Totali','S-Cumulativi','|V|','|E|','
            Spreader-Rate'])
    for x in range(0,ite):
        print "Saving data base file ..."
        RegistroColoriCoordVerdi() # reset lo scenario tutti i dischi tornano verdi

        CreaGrafoRandom() # prima creo la matrice di adiacenza
        CreaScenario() # dopo posso colorare i bianchi a caso, basandomi sulla matrice di
            adiacenza
        while S>0:
            ComputingRandomNoGUI()
            with open('RandomStat4b.csv', 'a') as csvfile:
                spamwriter = csv.writer(csvfile, delimiter=',', quotechar='|', quoting=csv.
                    QUOTE_MINIMAL)
                spamwriter.writerow([1,S,R,C,K,nodes,edges,float(K)/nodes])

def SalvaFileGrafoCompletoStatico():
    RegistroColoriCoordVerdi()
    CreaGrafoCompleto()
    CreaScenario()

    with open('CompleteStatic4b.csv', 'w') as csvfile:
        spamwriter = csv.writer(csvfile, delimiter=',', quotechar='|', quoting=csv.
            QUOTE_MINIMAL)
        spamwriter.writerow(['i','s','r','Azioni Totali','S-Cumulativi','|V|','|E|','
            Spreader-Rate'])

```

```

for x in range(0,ite):
    print "Saving data base file ..."
    RegistroColoriCoordVerdi()
    CreaGrafoCompleto()
    CreaScenario()
    while S>0:
        ComputingRandomNoGUI()
    with open('CompleteStatic4b.csv', 'a') as csvfile:
        spamwriter = csv.writer(csvfile, delimiter=',', quotechar='|', quoting=csv.
            QUOTE_MINIMAL)
        spamwriter.writerow([I,S,R,C,K,nodes,(nodes*nodes-nodes)*0.5,float(K)/nodes])

def CambiaMatrice(): # Modifica la matrice di adiacenza
    x=0
    global M
    while x<valore :

        i=random.sample(range(0,nodes),1)[0]
        j=random.sample(range(0,nodes),1)[0]
        if(i != j) :
            if( M[i,j]== 1):
                M[i,j]=M[j,i] = 0 # se c'è collegamento lo rompe
            elif( M[i,j] == 0 ):
                M[i,j]=M[j,i]=1 # se non c'è collegamento lo crea
        else:
            x=x-1 # se i=j il contatore torna indietro di un passo
        x=x+1

def SalvaFileVariabileNormale():
    RegistroColoriCoordVerdi()
    CreaGrafoRandom()
    CreaScenario()

    with open('RandomFlip4b.csv', 'w') as csvfile:
        spamwriter = csv.writer(csvfile, delimiter=',', quotechar='|', quoting=csv.
            QUOTE_MINIMAL)
        spamwriter.writerow(['i','s','r','Azioni Totali','S-Cumulativi','|V|','|E|','
            Spreader-Rate'])
    for x in range(0,ite):
        print "Saving data base file ..."
        RegistroColoriCoordVerdi()
        CreaGrafoRandom()
        CreaScenario()
        while S>0:
            if random.uniform(0,1) > 0.99 :
                CambiaMatrice()
                ComputingRandomNoGUI()
            with open('RandomFlip4b.csv', 'a') as csvfile:
                spamwriter = csv.writer(csvfile, delimiter=',', quotechar='|', quoting=csv.
                    QUOTE_MINIMAL)
                spamwriter.writerow([I,S,R,C,K,nodes,edges,float(K)/nodes])

def SalvaFileGrafoCompletoVariabile():
    RegistroColoriCoordVerdi()
    CreaGrafoCompleto()
    CreaScenario()
    with open('CompleteFlip4b.csv', 'w') as csvfile:
        spamwriter = csv.writer(csvfile, delimiter=',', quotechar='|', quoting=csv.
            QUOTE_MINIMAL)
        spamwriter.writerow(['i','s','r','Azioni Totali','S-Cumulativi','|V|','|E|','
            Spreader-Rate'])
    for x in range(0,ite):
        print "Saving data base file ..."
        RegistroColoriCoordVerdi()
        CreaGrafoCompleto()
        CreaScenario()
        while S>0:
            if random.uniform(0,1) > 0.5 :
                CambiaMatrice()
                ComputingRandomNoGUI()
            with open('CompleteFlip4b.csv', 'a') as csvfile:
                spamwriter = csv.writer(csvfile, delimiter=',', quotechar='|', quoting=csv.
                    QUOTE_MINIMAL)
                spamwriter.writerow([I,S,R,C,K,nodes,(nodes*nodes-nodes)*0.5,float(K)/nodes])

```

```

print "START RANDOM STAT"
SalvaFileStaticoNormale()
print "START RANDOM FLIP"
SalvaFileVariabileNormale()
print "START COMPLETE STAT"
SalvaFileGrafoCompletoStatico()
print "START COMPLETE FLIP"
SalvaFileGrafoCompletoVariabile()

```

Sorgente associato allo scenario con Stiflers pre esistenti:

```

# -*- coding: utf-8 -*- #

import math
import numpy
import random
import pygame
import csv

#### VALORI MODIFICABILI DALL'UTENTE ####
# Sia G = (V,E) grafo
nodes = 700 # |V| = nodes (vertici)
ite = 5 # numero di iterazioni
percentuale = 0.5 # percentuale degli archi modificati
#####

edges = random.sample(range(2*nodes, int((nodes*nodes-nodes)*0.5)), 1)[0] # |E| = edges (
    archi)
valore = int(edges*percentuale) # calcolo percentuale discretizzata
S = 0 # inizializzazione degli Spreaders
R = 0 # inizializzazione Stiflers
K = 1 # Spreaders cumulativi
C = 0 # cicli totali azioni compiute dagli Spreaders
SpreaderRate = K / nodes
rosso = 2 # il rosso nella grafica è associato agli Spreaders
verde = 1 # il verde nella grafica è associato agli Ignorants
bianco = 0 # il bianco nella grafica è associato agli Stiflers
RegistroColoriCoord = None

def SmeetI(): # (I,S,R) |- (I-1, S+1,R), transizione Spreader incontra Ignorant

    global I
    global S
    global C
    global K
    I=I-1
    S=S+1
    C=C+1
    K=K+1
    print "SPREADER INCONTRA IGNORANTE :",[I,S,R]

def SmeetS(): # (I,S,R) |- (I, S-2,R+2), transizione Spreader incontra Spreader

    global I
    global S
    global C
    global K
    global R
    I=I
    S=S-2
    R=R+2
    C=C+1
    print "SPREADER INCONTRA SPREADER :",[I,S,R]

def SmeetR(): # (I,S,R) |- (I, S-1,R+1), transizione Spreader incontra Stifler

    global I
    global S
    global C
    global K
    global R
    I=I
    S=S-1
    R=R+1
    C=C+1
    print "SPREADER INCONTRA SOFFOCATORE :",[I,S,R]

```

```

def CreaGrafoCompleto(): # crea una matrice di adiacenza per il grafo completo.
    global M
    global nodes
    M = numpy.zeros(shape=(nodes,nodes))
    for y1 in range(0,nodes):
        for y2 in range(0,nodes):
            M[y1,y2] = M[y2,y1]=1
            M[y1,y1]=0

def CreaGrafoRandom(): # crea una matrice di adiacenza randomizzata
    global M
    global nodes
    global edges
    x=0
    edges = random.sample(range(2*nodes,int((nodes*nodes-nodes)*0.5)),1)[0] # |E| = edges
    (archi)
    M = numpy.zeros(shape=(nodes,nodes))
    while x<edges :
        i = random.sample(range(0,nodes),1)[0] # associo ad i un valre {0,...,n-1}
        random
        j = random.sample(range(0,nodes),1)[0] # associo ad j un valre {0,...,n-1}
        random
        if(M[i,j] == 1) :
            x=x-1
        elif( M[i,j]== 0 & i!=j ) :
            M[i,j] = M[j,i]=1
        else:
            M[i,i] = 0
            x=x-1
    x=x+1

def RegistroColoriCoordVerdi(): # coloro tutto di verde, quindi resetto lo scenario per
una nuova iterazione
    global I
    global R
    global S
    global C
    global K
    global nodes
    global RegistroColoriCoord
    RegistroColoriCoord = numpy.zeros(shape=(nodes,3))
    for x in range(0,nodes) :
        i = (x/(2.*nodes))*math.cos(x) + 0.5
        j = (x/(2.*nodes))*math.sin(x) + 0.5
    # le coordinate verranno riproporzionate durante la Colorazione
    (RegistroColoriCoord[x,0])= verde
    (RegistroColoriCoord[x,1])=i
    (RegistroColoriCoord[x,2])=j

    S = 0
    I = nodes
    R = 0
    C = 0
    K = 1

def CreaScenario(): # vengono assegnati gli Stiflers iniziali ed uno Spreader a random
    global I
    global S
    global R
    global RegistroColoriCoord
    for x in range(0,nodes):
        if random.uniform(0,1) > 0.1 :
            RegistroColoriCoord[x,0]=bianco
            I = I-1
            R = R+1
    y = random.sample(range(0,nodes),1)[0]
    if RegistroColoriCoord[y,0] == verde :
        I = I-1
    if RegistroColoriCoord[y,0] == bianco :
        R = R-1
    RegistroColoriCoord[y,0]= rosso # Scelgo un rosso a caso.
    S=S+1

def ComputingRandomNoGUI() :
    global RegistroColoriCoord
    randspreadid=random.sample(numpy.where(RegistroColoriCoord[:,0]==rosso)[0],1)[0]
    # tra i contatti di randspreadid seleziono un nodo a caso connesso che chiamo z
    z = random.sample(numpy.where(M[randspreadid,:]==1)[0],1)[0]

    if RegistroColoriCoord[z,0]==verde:

```

```

        RegistroColoriCoord[z,0]=rosso
        Smeetl()

    elif RegistroColoriCoord[z,0]==bianco :
        RegistroColoriCoord[randspreaid,0]=bianco
        SmeetR()

    elif RegistroColoriCoord[z,0]==rosso :
        RegistroColoriCoord[randspreaid,0]=bianco
        RegistroColoriCoord[z,0]=bianco
        SmeetS()

def SalvaFileStaticoNormale():
    RegistroColoriCoordVerdi()
    CreaGrafoRandom()
    CreaScenario()

    with open('RandomStat.csv', 'w') as csvfile:
        spamwriter = csv.writer(csvfile, delimiter=',', quotechar='|', quoting=csv.
            QUOTE_MINIMAL)
        spamwriter.writerow(['i','s','r','Azioni Totali','S-Cumulativi','|V|','|E|','
            Spreader-Rate'])
    for x in range(0,ite):
        print "SalvaFile Grafo NORM STAT"
        RegistroColoriCoordVerdi() # reset lo scenario tutti i dischi tornano verdi

        CreaGrafoRandom() # prima creo la matrice di adiacenza
        CreaScenario() # dopo posso colorare i bianchi a caso, basandomi sulla matrice di
            adiacenza
        while S>0:
            ComputingRandomNoGUI()
            with open('RandomStat.csv', 'a') as csvfile:
                spamwriter = csv.writer(csvfile, delimiter=',', quotechar='|', quoting=csv.
                    QUOTE_MINIMAL)
                spamwriter.writerow([I,S,R,C,K,nodes,edges,float(K)/nodes])

def SalvaFileGrafoCompletoStatico():
    RegistroColoriCoordVerdi()
    CreaGrafoCompleto()
    CreaScenario()

    with open('GrafoCompletoStatico.csv', 'w') as csvfile:
        spamwriter = csv.writer(csvfile, delimiter=',', quotechar='|', quoting=csv.
            QUOTE_MINIMAL)
        spamwriter.writerow(['i','s','r','Azioni Totali','S-Cumulativi','|V|','|E|','
            Spreader-Rate'])
    for x in range(0,ite):
        print "SalvaFile Grafo Completo stat"
        RegistroColoriCoordVerdi()
        CreaGrafoCompleto()
        CreaScenario()
        while S>0:
            ComputingRandomNoGUI()
            with open('GrafoCompletoStatico.csv', 'a') as csvfile:
                spamwriter = csv.writer(csvfile, delimiter=',', quotechar='|', quoting=csv.
                    QUOTE_MINIMAL)
                spamwriter.writerow([I,S,R,C,K,nodes,(nodes*nodes-nodes)*0.5,float(K)/nodes])

def CambiaMatrice(): # Modifica la matrice di adiacenza
    x=0
    global M
    while x<valore :
        i=random.sample(range(0,nodes),1)[0]
        j=random.sample(range(0,nodes),1)[0]
        if(i!=j):
            if(M[i,j]==1):
                M[i,j]=M[j,i]=0 # se c'è collegamento lo rompe
            elif(M[i,j]==0):
                M[i,j]=M[j,i]=1 # se non c'è collegamento lo crea
        else:
            x=x-1 # se i=j il contatore torna indietro di un passo
    x=x+1

```

```

def SalvaFileVariabileNormale():
    RegistroColoriCoordVerdi()
    CreaGrafoRandom()
    CreaScenario()

    with open('RandomVar.csv', 'w') as csvfile:
        spamwriter = csv.writer(csvfile, delimiter=',', quotechar='|', quoting=csv.
            QUOTE_MINIMAL)
        spamwriter.writerow(['i','s','r','Azioni Totali','S-Cumulativi','|V|','|E|','
            Spreader-Rate'])
    for x in range(0,ite):
        print "SalvaFile Grafo NORMALE VARIABILE"
        RegistroColoriCoordVerdi()
        CreaGrafoRandom()
        CreaScenario()
        while S>0:
            if random.uniform(0,1) > 0.5 :
                CambiaMatrice()
                ComputingRandomNoGUI()
            with open('RandomVar.csv', 'a') as csvfile:
                spamwriter = csv.writer(csvfile, delimiter=',', quotechar='|', quoting=csv.
                    QUOTE_MINIMAL)
                spamwriter.writerow(['I,S,R,C,K,nodes,float(K)/nodes'])

def SalvaFileGrafoCompletoVariabile():
    RegistroColoriCoordVerdi()
    CreaGrafoCompleto()
    CreaScenario()
    with open('GrafoCompletoVar.csv', 'w') as csvfile:
        spamwriter = csv.writer(csvfile, delimiter=',', quotechar='|', quoting=csv.
            QUOTE_MINIMAL)
        spamwriter.writerow(['i','s','r','Azioni Totali','S-Cumulativi','|V|','|E|','
            Spreader-Rate'])
    for x in range(0,ite):
        print "SalvaFile Grafo Completo VARIABILE"
        RegistroColoriCoordVerdi()
        CreaGrafoCompleto()
        CreaScenario()
        while S>0:
            if random.uniform(0,1) > 0.5 :
                CambiaMatrice()
                ComputingRandomNoGUI()

            with open('GrafoCompletoVar.csv', 'a') as csvfile:
                spamwriter = csv.writer(csvfile, delimiter=',', quotechar='|', quoting=csv.
                    QUOTE_MINIMAL)
                spamwriter.writerow(['I,S,R,C,K,nodes,(nodes*nodes-nodes)*0.5,float(K)/nodes'])

print "INIZIO ALGORITMO 1"
SalvaFileStaticoNormale()
print "INIZIO ALGORITMO 2"
SalvaFileGrafoCompletoStatico()
print "INIZIO ALGORITMO 3"
SalvaFileGrafoCompletoVariabile()
print "INIZIO ALGORITMO 4"
SalvaFileVariabileNormale()

```

Qui di seguito vengono riportati i metodi per il render grafico *Colorazione()* e *DisegnaLati()* della matrice di adiacenza. Per non appesantire il programma sono state tolte. Ma andando ad operare in modo opportuno sul codice precedente possono essere reintrodotte.

```

def Colorazione(): # Colorazione dei nodi
    for x in range(0,nodes): # scansione da 0 a I+S+R-1

        if RegistroColoriCoord[x,0] == verde :
            a=RegistroColoriCoord[x,1] # a assume valore della seconda colonna
            di RegistroColoriCoord
            b=RegistroColoriCoord[x,2] # b assume valore della terza colonna
            di RegistroColoriCoord
            pygame.draw.circle(screen,(0,255,0),(int(a*1024),int(b*800)),3) #
                vengono disegnati dei dischi, la terna colori è RGB

        elif RegistroColoriCoord[x,0] == rosso :

```



```

        a=RegistroColoriCoord[x,1]
        b=RegistroColoriCoord[x,2]
        pygame.draw.circle(screen,(255,0,0),(int(a*1024),int(b*800)),3)
    elif RegistroColoriCoord[x,0] == bianco :
        a=RegistroColoriCoord[x,1]
        b=RegistroColoriCoord[x,2]

        pygame.draw.circle(screen,(255,255,255),(int(a*1024),int(b*800)),
        ,3)

    pygame.display.flip() # aggiorna la schermata

def DisegnaLati(): # disegna i segmenti che connettono i nodi
    for x in range(0,nodes) :
        for y in range(0,nodes) : # Prendo le coordinate x,y dei nodi con
            connessioni e li passo alla grafica
            if M[x,y]==1 :
                v1 = RegistroColoriCoord[x,1] # NODO Y,X
                w1= RegistroColoriCoord[x,2]
                v2 = RegistroColoriCoord[y,1] # NODO X,Y
                w2= RegistroColoriCoord[y,2]
                pygame.draw.line(screen, (0, 0, 130), (v1*1024, w1*800), (v2
                *1024, w2*800)) #espando i punti da normalizzati

    pygame.display.flip() # aggiorna la schermata a fine ciclo for

def CreaFinestra(): #viene creata una finestra
    screen = pygame.display.set_mode((1024,800)) # genera una finestra 1024 per 800
    screen.fill((0, 0, 0) # lo sfondo della finestra viene riempito di nero
    pygame.display.flip() # lo schermo viene aggiornato

```

Per un miglioramento operativo, come già detto, sono state rimosse le funzioni grafiche: Colorazione(), DisegnaLati() e CreaFinestra(). D'altra parte al fine operativo non hanno una rilevanza, si tratta solamente di un fatto di estetica. Nelle figure 8.1 e 8.2 riportiamo un breve esempio di come opera la procedura precedentemente argomentata su di un piccolo grafo $G = (V, E)$ con $|V| = 5$ nodi e $|E| = 4$ archi. la Figura 8.3 rappresenta il RegistroColoriCoord all'inizio della procedura dopo che viene inizializzato il primo Spreader. Nelle Figure 8.4 e 8.5 possiamo vedere due screenshot del programma in esecuzione.

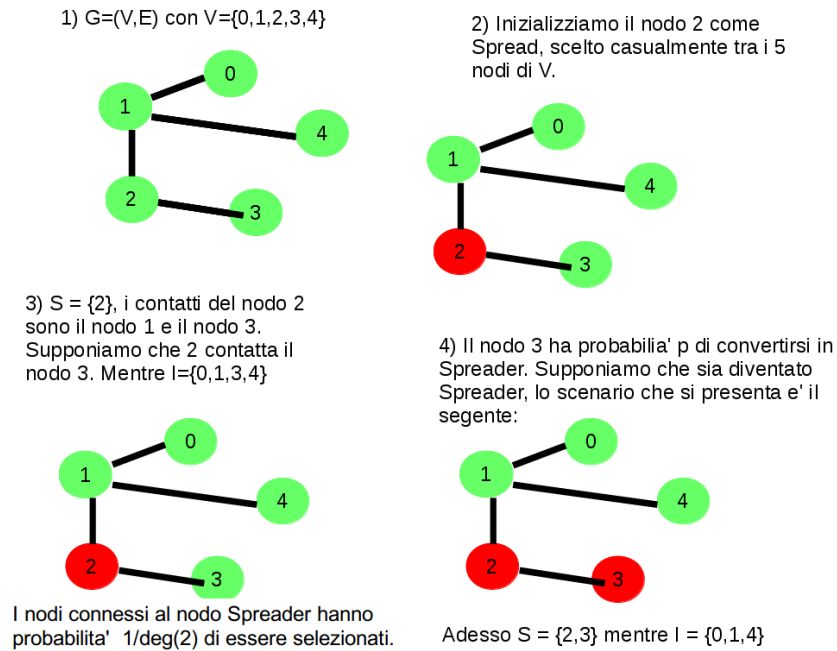
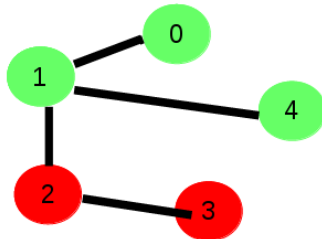


Figura 8.1: Un esempio del programma su un piccolo grafo.

- 5) Con probabilita' $1/|S|$ viene scelto un nodo tra 2 e 3 di S , supponiamo venga scelto il nodo 3. Questo viene Forzato a contattare il nodo 2 suo unico contatto



- 6) Il nodo 2 e il nodo 3 si convertono in Stifler. Adesso S risulta essere vuoto, quindi la procedura termina con gli insiemi $I=\{0,1,4\}$ e $R=\{2,3\}$ e la terna finale $(i, \theta, r) = (3, 0, 2)$

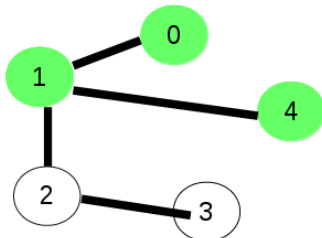


Figura 8.2: Un esempio del programma su un piccolo grafo.

| COLOR | Coordinate X | Coordinate Y |
|-------|------------------------------|------------------------------|
| GREEN | 0.5 | 0.5 |
| GREEN | $(1/(2 V)) * \cos(1) + 0.5$ | $(1/(2 V)) * \sin(1) + 0.5$ |
| RED | $(2/(2 V)) * \cos(2) + 0.5$ | $(2/(2 V)) * \sin(2) + 0.5$ |
| GREEN | $(3/(2 V)) * \cos(3) + 0.5$ | $(3/(2 V)) * \sin(3) + 0.5$ |
| GREEN | $(4/(2 V)) * \cos(4) + 0.5$ | $(4/(2 V)) * \sin(4) + 0.5$ |

Figura 8.3: Tabella di RegistroColoriCoord() dopo che viene inizializzato lo Spreader iniziale nella figura 8.1.

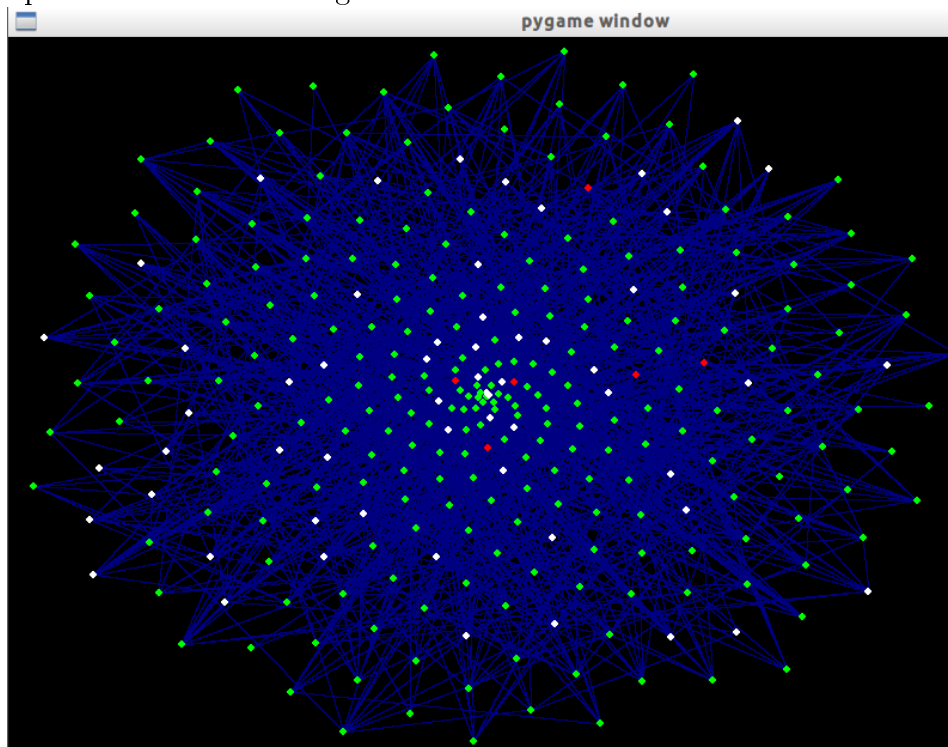
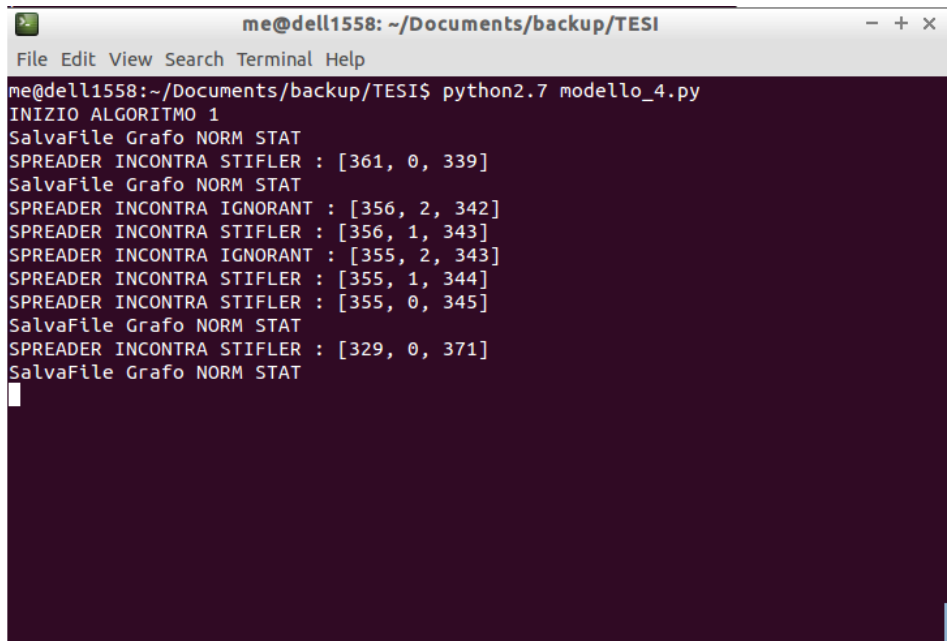


Figura 8.4: Screenshot del programma con aggiunta di CreaFinestra(), Colorazione() e DisegnaLati().



```
me@dell1558: ~/Documents/backup/TESt
File Edit View Search Terminal Help
me@dell1558:~/Documents/backup/TESt$ python2.7 modello_4.py
INIZIO ALGORITMO 1
SalvaFile Grafo NORM STAT
SPREADER INCONTRA STIFLER : [361, 0, 339]
SalvaFile Grafo NORM STAT
SPREADER INCONTRA IGNORANT : [356, 2, 342]
SPREADER INCONTRA STIFLER : [356, 1, 343]
SPREADER INCONTRA IGNORANT : [355, 2, 343]
SPREADER INCONTRA STIFLER : [355, 1, 344]
SPREADER INCONTRA STIFLER : [355, 0, 345]
SalvaFile Grafo NORM STAT
SPREADER INCONTRA STIFLER : [329, 0, 371]
SalvaFile Grafo NORM STAT
```

Figura 8.5: Screenshot del programma senza la grafica, con gli output parziali delle transizioni; versione con gli Stiflers pre esistenti nello scenario.