Instituto Politécnico Nacional

Centro de Investigación en Computación

Análisis de un modelo de redes de pares-a-pares bajo un ataque cibernético utilizando cadenas de Markov

Natalia Sánchez Patiño
Dra. Gina Gallegos García
Dr. Mario Rivero Ángeles

22 de septiembre 2021

Índice

1. Teoría

1.1. Cadena simple de Markov

Esta primera cadena simple, simula el comportamiento de una red de pares a pares donde existen dos tipos de usuarios, sanguijuelas y semillas.

Para el desarrollo de este modelo se toman en cuenta dos escenarios de condiciones; Penuria, donde hay escasez de usuarios y el comportamiento esta dato por la ecuación $M(Nx + y)$, donde M significa la tasa de subida de archivos, N la tasa de bajada, x los usuarios con trozos de archivos y y usuarios con los archivos completos a los que es más probable enviar peticiones para la descarga de sus archivos. Y la Abundancia, donde aumenta la cantidad de usuarios y por lo tanto también los recursos, representándose con la ecuación Cx, siendo C la tasa de baja de archivos en estos casos.

Otras variables a tomar en cuenta son la tasa de arribos (L), es decir cuántos usuarios llegan por segundo y los tiempos de conexión de los usuarios H para las sanguijuelas y G para las semillas.

Esta serie de variables se relacionan utilizando las siguientes ecuaciones:

$$\tau = \min(M(Nx + y), Cx)$$

$$T_1 = -\frac{1}{L}\log(1-u)$$

$$T_2 = -\frac{1}{Hx}\log(1-u)$$

$$T_3 = -\frac{1}{Gy}\log(1-u)$$

$$T_4 = -\frac{1}{\tau}\log(1-u)$$

$$T = \min(T_1, T_2, T_3, T_4)$$

Cada una de estas ecuaciones representa un cambio de estado en la cadena, cada uno de ellos recibe un valor en cada iteración realizada y aquel con el valor mínimo es el que determina a qué estado se hace la transición. Estos cambios son guardados dentro de una matriz para saber cuánto tiempo se permanece dentro de un estado. Algo importante para generar este cambio de estados es un número semi-aleatorio entre cero y uno con distribución uniforme que influirá en el peso que tendrán estas ecuaciones a cada paso.

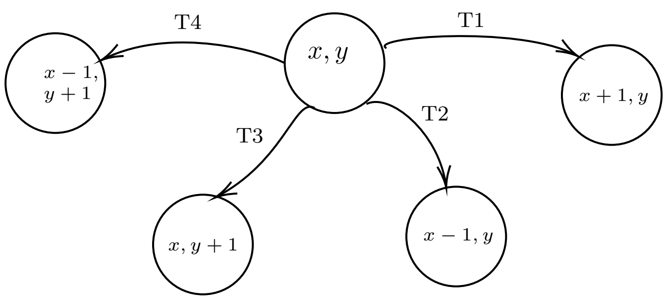

Figura 1: Modelado utilizando una cadena simple de Markov

Esta cadena es una cadena de Markov irreducible porque desde cualquier estado se puede acceder a cualquier otro y todos los estados se comunican entre sí. Los sanguijuelas son usuarios que tienen partes de archivos o ningún dato y los sedes son aquellos pares que ya han descargado el archivo completo y esperan en el sistema para compartir sus recursos. Ambos cooperan para llevar archivos a otros leeches o sanguijuelas. Entre más usuarios con archivos completos se encuentren dentro de la red, los tiempos de descarga serán más ágiles permitiendo que el tráfico y el ancho de banda aumenten. Esto sin embargo debe de alcanzar un límite en algún punto, que es cuando el tráfico alcanza la estabilidad y la red se mantiene en movimiento sin grandes cambios en el número de usuarios y es el tipo de comportamiento que se desea obtener.

1.2. Cadena de Markov con nodos iniciales infectados

$$X_n \rightarrow \text{sanguijuelas no infectadas}$$
$$X_n \rightarrow \text{sanguijuelas infectadas}$$
$$Y_n \rightarrow \text{semillas no infectadas}$$
$$Y_i \rightarrow \text{semillas infectadas}$$

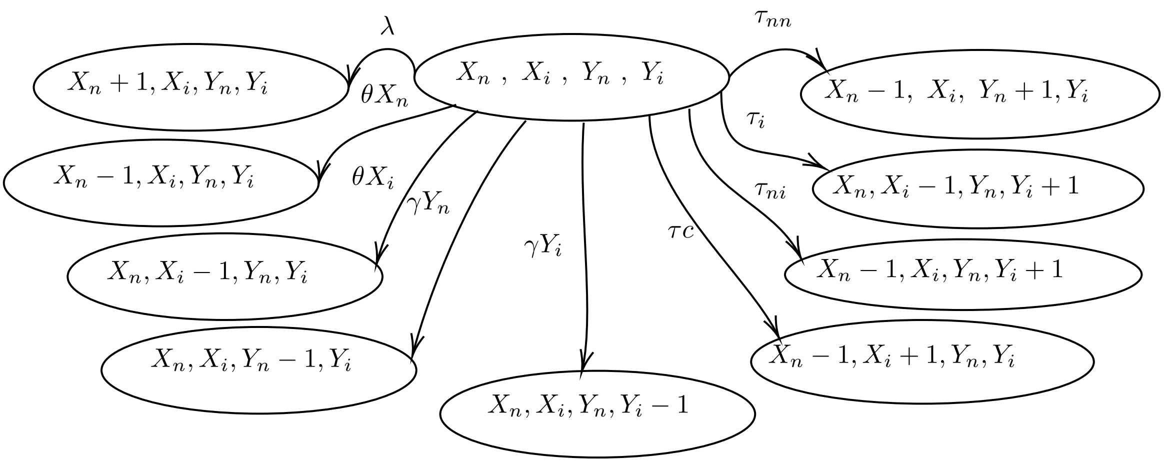

Figura 2: Diagrama de los posibles estados de la cadena infectada

1. Las sanguijuelas no están infectados
2. Si un par entra en contacto con otro par infectado se infecta. Tasa de infección = 1.
3. $X_i \geq N_i \rightarrow$ Debe haber al menos un par malicioso
4. $N_i \rightarrow$ número inicial de nodos maliciosos
5. No hay transiciones de $X_i \rightarrow X_n$ ya que si está infectado, ya no se puede desinfectar.
6. El nodo ya está infectado $\rightarrow$ no importa de donde descarga, se pasa a semilla infectada.

$$X = X_n + X_i$$
$$Y = Y_n + Y_i$$
$$\tau_i = \text{máx}\left\{ CX_i, (X\eta + Y)\,\frac{X_i}{X} \right\}$$

7. Sólo los pares (sanguijuelas y semillas) infectados.
8. Si el peer ya está infectado $\rightarrow$ se mantiene infectado.
9. $\tau_{nn} \rightarrow$ Indica si el par actual no se infectaría.
10. $\tau_{ni} \rightarrow$ Indica la descarga de los pares, pero al menos uno de ellos debe de estar contaminado.

$$\tau_{nn} = \text{mín}\left\{CP_nX_n, \mu\left(X_n\eta + Y_n\right)\right\}$$
$$\tau_{ni} = \text{mín}\left\{C(1-P_n)X_i, \mu\left(X_i\eta + Y_n\right)(1-P_n)\right\}$$

11. $P_n \rightarrow$ Probabilidad de no infectarse
12. El ancho de banda total es: $\mu(X\eta + Y)$
13. El ancho de banda infectado es: $\mu(X_i\eta + Y_i)$
14. El ancho de banda total es: $\mu(X_n\eta + Y_n)$

$$\mu X_i\eta + \mu Y_i + \mu X_n\eta + \mu Y_n$$
$$= \mu\left[\eta(X_i + X_n) + Y_i + Y_n\right] = \mu\left[\eta X + Y\right]$$
$$\Rightarrow P_n = \frac{\mu(X_n\eta + Y_n)}{\mu(X\eta + Y)} = \frac{X_n\eta + Y_n}{X\eta + Y}$$

El número de pares infectados con los que se puede conectar:

$$\eta X_i + Y_i$$

El número de pares no infectados con los que se puede conectar:

$$\eta X_n + Y_n$$

La forma de conectarse con nodos no infectados:

$$P_n = \frac{X_n\eta + Y_n}{X\eta + Y}$$

13. $\tau_C \rightarrow$ Tasa con la que una sanguijuela se contamina.
    Se puede contaminar cada pedazo descargado. El archivo es de tamaño F (F: 1 normalizado). El tamaño del pedazo es B. El archivo tiene $K = \frac{F}{B}$ chuks.

$$\tau_C = \text{mín}\left\{CK(1-P_n)X_n, \mu K(\eta X + Y)(1-P_n)\right\}$$

14. $CK, \mu K \rightarrow$ La tasa de descarga de un pedazo es K veces más rápida que la del archivo total.
15. $1 - P_n \rightarrow$ Probabilidad de infectarse

Estas operaciones determinan el comportamiento de los pares o usuarios dentro de la red pues determinan la probabilidad de cambiar de estado según las condiciones que se tengan en ese momento determinado.

1.3. Cadena de Markov con nodos iniciales infectados y un parámetro como contramedida

$P_I \rightarrow$ Probabilidad de que un nodo NO infectado que descarga de un nodo Infectado SÍ se contamine.

Este nuevo parámetro que se agrega a esta última aproximación representa aquellas medidas preventivas que se pueden tomar para que cuando un nodo sano entre en contacto con un nodo malicioso la probabilidad de contagio no sea del 100 %. Un ejemplo de esta clase de medidas pueden ser los antivirus o los protocolos de verificación de procedencia del archivo o protocolos de cuarentena de archivos de los que no se sabe si se pueden confiar.

Mientras más cercano a 1 se halle este parámetro mayor será la probabilidad de contagio, y si se encuentra más cerca del cero, esto indica que las medidas asumidas son de mayor efectividad.

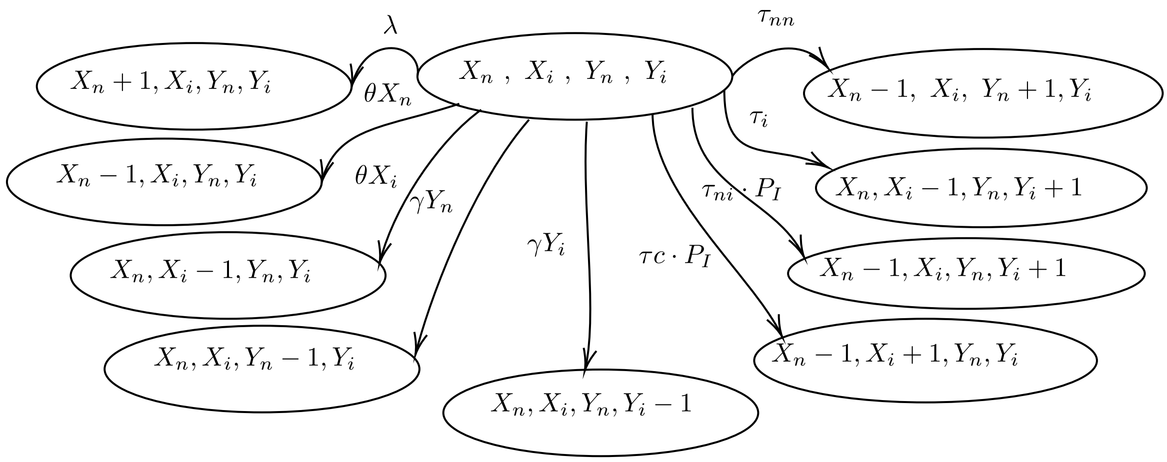

Figura 3: Diagrama de los posibles estados de la cadena con un factor de contramedida al ataque cibernético

2. Implementación

2.1. Materiales

- Python 3.6
- Librerías
    - Math
    - Numpy
    - Matplotlib
    - Random
    - Pandas

2.2. Código

Para las tres implementaciones se utilizo una función generadora de números aleatorios del 0 al 100,000.

Listing 1: Librerías y función de números aleatorios

```
import random
import math
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

def u():
  u = random.uniform(0,1000000)/1000000
  return u
```

El código utilizado para la cadena simple fue el siguiente:
Se utiliza una serie de condiciones if para evitar que se de una división entre 0, esto corresponderá a los usuarios que estén dentro de la red en ese momento.

Listing 2: Cadena con infectada

```
# Estado inicial

i = 0 # Peers Leeches
x = 1 #Peers Seeds
```

```
# Parmetros

C = 0.02
L = 1
M = 0.00125
H = 0.01
G = 0.01
N = 0.85
O = 1
P = 1

# Inicializar listas, arrays y algunas variables
TCsim = 0 # Almacenar el tiempo de simulacin
pi = Array{Float64}(undef, 200,200) #Almacena el tiempo en que paso por determinado estado
I = [] # Guarda en cada iteracin el valor actual de i
X2 = [] # Guarda en cada iteracin el valor actual de x
num = [] #Guarda una lista con los nmeros de iteraciones
T1,T2,T3,T4,T5,T6,T7,T8,T9, T10 = Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing,
    Nothing, Nothing, Nothing

# Ciclo de iteraciones de 1 a 100,000 con un paso de uno

for j in 1:1:100000
    append!(num,j)
    if x == 0
        x = 1
    end
    if i == 0 && x == 1
        T = T1 = -(1/L)*log(1-u())
    end
    if i == 0
        T1 = -(1/L)*log(1-u())
        T3 = -(1/(G*x))*log(1-u())
        T = min(T1,T3)
    elseif x == 1
        tau = min(M*(N*i+x),C*i)
        T1 = -(1/L)*log(1-u())
        T2 = -(1/(H*i))*log(1-u())
        T4 = -(1/tau)*log(1-u())
        T = min(T1,T2,T4)
    else
        tau = min(M*(N*i+x),C*i)
        T1 = -(1/L)*log(1-u())
        T2 = -(1/(H*i))*log(1-u())
        T3 = -(1/(G*x))*log(1-u())
        T4 = -(1/tau)*log(1-u())
        T5 = -(1/P)*log(1-u())
        T6 = -(1/O)*log(1-u())
        T = min(T1,T2,T3,T4,T5,T6)
    end

    # Una vez que se obtuvo el tiempo minimo se encuentra
    # al estado que pertenece y conforme a ese estado se
    # modifica ya sea los seeds, leeches o ambos.

    if T == T1
        pi[i+1,x] += T
        i += 1
    elseif T == T2
```

```
        pi[i-1,x] += T
        i -= 1
    elseif T == T3
        pi[i-1,x-1] += T
        i -= 1
        x -= 1
    elseif T == T4
        pi[i+1,x+1] += T
        i += 1
        x += 1
    elseif T == T5
        pi[i,x] += T
    elseif T == T6
        pi[i+1,x+1] += T
        i += 1
        x += 1
    end
    append!(I,i)
    append!(X2,x)
    println(i, '\t', x)
    TCsim += T
end

# Se grafican los resultados obtenidos
p1 = plot(num,I, label= :none, title="Seeds")
p2 = plot(num,X2, color = :green, label= :none, title="Leeches")
plot(p1, p2, layout = (2, 1))
```

Los resultados que se esperan de este archivo es una imagen con las gráficas de la evolución en cada iteración de los usuarios para esta cadena, la primera muestra los seeds y la segunda los leechers.

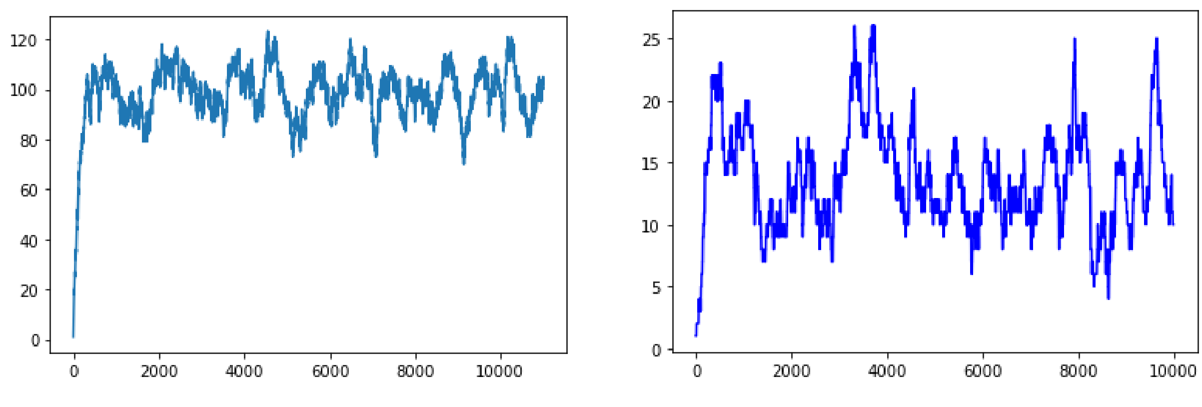

Figura 4: Resultados cadena simple

La segunda implementación se sigue basando en la primera y lo que sucede es únicamente aumentar los estados como casos

Listing 3: Cadena con infectada

```
# Nodos infectados
Ni = 1

# Estado inicial

xn = 0
yn = 1
```

```
xi = Ni
yi = 1

#  Parmetros

C=0.002
L = 2
M = 0.05
H = 0.005
G = .3
N = 0.85
K = 1/10

Tsim = 0
pix = Array{Float64}(undef, 5000,6000)
piy = Array{Float64}(undef, 8000,6000)
Xn = []
Yn = []
Xi = []
Yi = []
num = []
S1,S2,S3,S4,S5,S6,S7,S8,S9,S10=0,0,0,0,0,0,0,0,0,0
T1,T2,T3,T4,T5,T6,T7,T8,T9,T10 = Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing,
    Nothing, Nothing, Nothing

for i in 1:100000
    append!(num,i)
    if yn == 0
        yn = 1
    end
    if yi == 0
        yi = 1
    end
    if xn == 0 && yn == 1
        T = T1 = -(1/L)*log(1-u())
    elseif xn == 0
        T1 = -(1/L)*log(1-u())
        T3 = -(1/(H*xi))*log(1-u())
        T = min(T1,T3)
    elseif yn == 1
        Pn = (xn*M+yn)/(xn+(yn+yi))
        tauni = min(C*(1-Pn)*xi,M*(N*xn+yn)*(1-Pn))
        taunn = min(C*Pn*xn,M*(N*xn+yn))
        T1 = -(1/L)*log(1-u())
        T2 = -(1/(H*xn))*log(1-u())
        T4 = -(1/G*yn)*log(1-u())
        T9 = -(1/taunn)*log(1-u())
        T = min(T1,T2,T4,T9)
    elseif xi != 0 && yi < yn
        Pn = (xn*M+yn)/(xn+(yn+yi))
        tauni = min(C*(1-Pn)*xi,M*(N*xn+yn)*(1-Pn))
        T7 = -(1/tauni)*log(1-u())
        T = T7
    elseif xi == 0 && yi == 1
        Pn = (xn*M+yn)/(xn+(yn+yi))
        tauni = min(C*(1-Pn)*xi,M*(N*xn+yn)*(1-Pn))
        taunn = min(C*Pn*xn,M*(N*xn+yn))
        taui = max(C*xi,((xn+xi)*N+(yn+yi)*(xi/(xn+xi))))
        tauC = min(C*K*(1-Pn)*xn,M*K*(M*(xn+xi)+(yn+yi))*(1-Pn))
        T1 = -(1/L)*log(1-u())
```

```
        T2 = -(1/(H*xn))*log(1-u())
        T4 = -(1/G*yn)*log(1-u())
        T6 = -(1/(tauC))*log(1-u())
        T9 = -(1/taunn)*log(1-u())
        T = min(T1,T2,T4,T6,T9)
    elseif xi == 0
        Pn = (xn*M+yn)/(xn+(yn+yi))
        tauni = min(C*(1-Pn)*xi,M*(N*xn+yn)*(1-Pn))
        taunn = min(C*Pn*xn,M*(N*xn+yn))
        taui = max(C*xi,((xn+xi)*N+(yn+yi)*(xi/(xn+xi))))
        tauC = min(C*K*(1-Pn)*xn,M*K*(M*(xn+xi)+(yn+yi))*(1-Pn))
        T1 = -(1/L)*log(1-u())
        T2 = -(1/(H*xn))*log(1-u())
        T4 = -(1/G*yn)*log(1-u())
        T5 = -(1/(G*yi))*log(1-u())
        T6 = -(1/(tauC))*log(1-u())
        T9 = -(1/taunn)*log(1-u())
        T = min(T1,T2,T4,T5,T6,T9)
    else
        Pn = (xn*M+yn)/(xn+(yn+yi))
        tauni = min(C*(1-Pn)*xi,M*(N*xn+yn)*(1-Pn))
        taunn = min(C*Pn*xn,M*(N*xn+yn))
        taui = min(C*xi,((xn+xi)*N+(yn+yi)*(xi/(xn+xi))))
        tauC = min(C*K*(1-Pn)*xn,M*K*(M*(xn+xi)+(yn+yi))*(1-Pn))
        T1 = -(1/L)*log(1-u())
        T2 = -(1/(H*xn))*log(1-u())
        T3 = -(1/(H*xi))*log(1-u())
        T4 = -(1/G*yn)*log(1-u())
        T5 = -(1/(G*yi))*log(1-u())
        T6 = -(1/(tauC))*log(1-u())
        T7 = -(1/tauni)*log(1-u())
        T8 = -(1/taui)*log(1-u())
        T9 = -(1/taunn)*log(1-u())
        T = min(T1,T2,T3,T4,T5,T6,T7,T8,T9)
    end

    if T == T1
        pix[1+xn+1,1+xi] += T
        piy[1+yn,1+yi] += T
        xn += 1
        S1 += 1
    elseif T == T2
        pix[1+xn-1,1+xi] += T
        piy[1+yn,1+yi] += T
        xn -= 1
        S2 += 1
    elseif T == T3
        pix[1+xn,1+xi-1]
        piy[1+yn,1+yi] += T
        xi -= 1
        S3 += 1
    elseif T == T4
        pix[1+xn,1+xi] += T
        piy[1+yn-1,1+yi] += T
        yn -= 1
        S4 += 1
    elseif T == T5
        pix[1+xn,1+xi] += T
        piy[1+yn,1+yi-1] += T
        yi -= 1
```

```
            S5 += 1
        elseif T == T6
            pix[1+xn-1,1+xi+1] += T
            piy[1+yn,1+yi] += T
            xn -= 1
            xi += 1
            S6 += 1
        elseif T == T7
            pix[1+xn-1,1+xi] += T
            piy[1+yn,1+yi+1] += T
            xn -= 1
            yi += 1
            S7 += 1
        elseif T == T8
            pix[1+xn,1+xi-1] += T
            piy[1+yn,1+yi+1] += T
            xi -= 1
            yi += 1
            S8 += 1
        elseif T == T9
            pix[1+xn-1,1+xi] += T
            piy[1+yn+1,1+yi] += T
            xn -= 1
            yn += 1
            S9 += 1
        elseif T == T10
            pix[1+xn,1+xi] += T
            piy[1+yn+1,1+yi] += T
            yn += 1
            S10 += 1
        end
        append!(Xn,xn)
        append!(Yn,yn)
        append!(Xi,xi)
        append!(Yi,yi)
        T1,T2,T3,T4,T5,T6,T7,T8,T9, T10 = Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing,
            Nothing, Nothing, Nothing
        Tsim += T
    end
print(S1,'\t',S2,'\t',S3,'\t',S4,'\t',S5,'\t',S6,'\t',S7,'\t',S8,'\t',S9,'\t',S10)
TXn = pd.Series(Xn)
TYn = pd.Series(Yn)
TXi = pd.Series(Xi)
TYi = pd.Series(Yi)

fig, axs = plt.subplots(2, 2, figsize=(10,8))
axs[0, 0].plot(TXn.index,TXn.values)
axs[0, 0].axhline(y=sum(Xn)/100000,c='k')
axs[0, 0].set_title('Leeches sanos')
axs[0, 1].plot(TYn.index,TYn.values, 'tab:orange')
axs[0, 1].axhline(y=sum(Yn)/100000,c='k')
axs[0, 1].set_title('Seeds sanos')
axs[1, 0].plot(TXi.index,TXi.values, 'tab:green')
axs[1, 0].axhline(y=sum(Xi)/100000,c='k')
axs[1, 0].set_title('Leeches infectados')
axs[1, 1].plot(TYi.index,TYi.values, 'tab:red')
axs[1, 1].axhline(y=sum(Yi)/100000,c='k')
axs[1, 1].set_title('Seeds infectados')
fig.tight_layout()
print('Leeches sanos:',sum(Xn)/100000)
```

```
print('Seeds sanos:',sum(Yn)/100000)
print('Leeches infectados:',sum(Xi)/100000)
print('Seeds infectados:',sum(Yi)/100000)
print('Valores usados: C ',C,'L',L,'M',M,'H',H,'G',G,'N',N,'K',K)
```

Los resultados esperados son algo similar a la figura 5

```
Leeches sanos: 335.19332
Seeds sanos: 1.72068
Leeches infectados: 9.4831
Seeds infectados: 2.3969
Valores usados: C  0.002 L 2 M 0.05 H 0.005 G 0.3 N 0.85 K 0.1
```

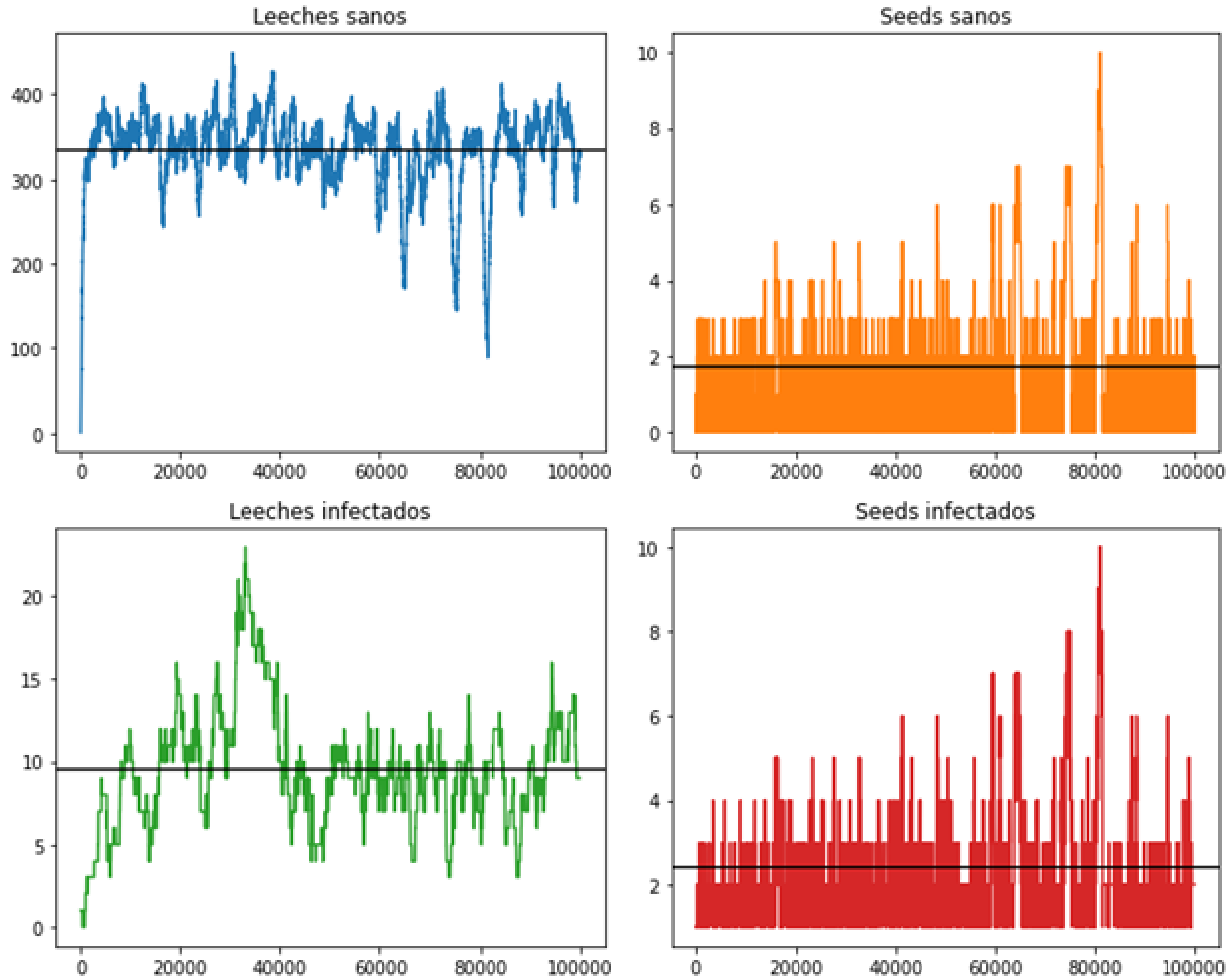

Figura 5: Resultados cadena infectada

La tercera implementación aumenta un parámetro a dos de los estados y el código es muy similar al anterior.

Listing 4: Cadena con contramedidas

```
# Nodos infectados
Ni = 1

# Estado inicial

xn = 0
```

```
yn = 1
xi = Ni
yi = 1

# Parmetros

C=0.002
L = 2
M = 0.05
H = 0.005
G = .3
N = 0.85
K = 1/10

Tsim = 0
pix = np.zeros([5000,6000])
piy = np.zeros([8000,6000])
Xn = []
Yn = []
Xi = []
Yi = []
num = []
S1,S2,S3,S4,S5,S6,S7,S8,S9,S10=0,0,0,0,0,0,0,0,0,0
T1,T2,T3,T4,T5,T6,T7,T8,T9, T10 = None, None, None, None, None, None, None, None, None, None

for i in range(100000):
  num.append(i)
  if yn == 0:
    yn = 1
  if yi == 0:
    yi = 1
  if xn == 0 and yn == 1:
    T = T1 = -(1/L)*math.log(1-u())
  elif xn == 0:
    T1 = -(1/L)*math.log(1-u())
    T3 = -(1/(H*xi))*math.log(1-u())
    T = min(T1,T3)
  elif yn == 1:
    Pn = (xn*M+yn)/(xn+(yn+yi))
    tauni = min(C*(1-Pn)*xi,M*(N*xn+yn)*(1-Pn))
    taunn = min(C*Pn*xn,M*(N*xn+yn))
    T1 = -(1/L)*math.log(1-u())
    T2 = -(1/(H*xn))*math.log(1-u())
    T4 = -(1/G*yn)*math.log(1-u())
    T9 = -(1/taunn)*math.log(1-u())
    T = min(T1,T2,T4,T9)
  elif xi != 0 and yi < yn:
    tauni = min(C*(1-Pn)*xi,M*(N*xn+yn)*(1-Pn))
    T7 = -(1/tauni)*math.log(1-u())
    T = T7
  elif xi == 0 and yi == 1:
    Pn = (xn*M+yn)/(xn+(yn+yi))
    tauni = min(C*(1-Pn)*xi,M*(N*xn+yn)*(1-Pn))
    taunn = min(C*Pn*xn,M*(N*xn+yn))
    taui = max(C*xi,((xn+xi)*N+(yn+yi)*(xi/(xn+xi))))
    tauC = min(C*K*(1-Pn)*xn,M*K*(M*(xn+xi)+(yn+yi))*(1-Pn))
    T1 = -(1/L)*math.log(1-u())
    T2 = -(1/(H*xn))*math.log(1-u())
    T4 = -(1/G*yn)*math.log(1-u())
    T6 = -(1/(tauC))*math.log(1-u())
```

```
        T9 = -(1/taunn)*math.log(1-u())
        T = min(T1,T2,T4,T6,T9)
    elif xi == 0:
        Pn = (xn*M+yn)/(xn+(yn+yi))
        tauni = min(C*(1-Pn)*xi,M*(N*xn+yn)*(1-Pn))
        taunn = min(C*Pn*xn,M*(N*xn+yn))
        taui = max(C*xi,((xn+xi)*N+(yn+yi)*(xi/(xn+xi))))
        tauC = min(C*K*(1-Pn)*xn,M*K*(M*(xn+xi)+(yn+yi))*(1-Pn))
        T1 = -(1/L)*math.log(1-u())
        T2 = -(1/(H*xn))*math.log(1-u())
        T4 = -(1/G*yn)*math.log(1-u())
        T5 = -(1/(G*yi))*math.log(1-u())
        T6 = -(1/(tauC))*math.log(1-u())
        T9 = -(1/taunn)*math.log(1-u())
        T = min(T1,T2,T4,T5,T6,T9)
    else:
        Pn = (xn*M+yn)/(xn+(yn+yi))
        tauni = min(C*(1-Pn)*xi,M*(N*xn+yn)*(1-Pn))
        taunn = min(C*Pn*xn,M*(N*xn+yn))
        taui = min(C*xi,((xn+xi)*N+(yn+yi)*(xi/(xn+xi))))
        tauC = min(C*K*(1-Pn)*xn,M*K*(M*(xn+xi)+(yn+yi))*(1-Pn))
        T1 = -(1/L)*math.log(1-u())
        T2 = -(1/(H*xn))*math.log(1-u())
        T3 = -(1/(H*xi))*math.log(1-u())
        T4 = -(1/G*yn)*math.log(1-u())
        T5 = -(1/(G*yi))*math.log(1-u())
        T6 = -(1/(tauC))*math.log(1-u())
        T7 = -(1/tauni)*math.log(1-u())
        T8 = -(1/taui)*math.log(1-u())
        T9 = -(1/taunn)*math.log(1-u())
        T = min(T1,T2,T3,T4,T5,T6,T7,T8,T9)

    if T == T1:
        pix[xn+1][xi] += T
        piy[yn][yi] += T
        xn += 1
        S1 += 1
    elif T == T2:
        pix[xn-1][xi]
        piy[yn][yi] += T
        xn -= 1
        S2 += 1
    elif T == T3:
        pix[xn][xi-1]
        piy[yn][yi] += T
        xi -= 1
        S3 += 1
    elif T == T4:
        pix[xn][xi] += T
        piy[yn-1][yi] += T
        yn -= 1
        S4 += 1
    elif T == T5:
        pix[xn][xi] += T
        piy[yn][yi-1] += T
        yi -= 1
        S5 += 1
    elif T == T6:
        pix[xn-1][xi+1] += T
        piy[yn][yi] += T
```

```
    xn -= 1
    xi += 1
    S6 += 1
  elif T == T7:
    pix[xn-1][xi] += T
    piy[yn][yi+1] += T
    xn -= 1
    yi += 1
    S7 += 1
  elif T == T8:
    pix[xn][xi-1] += T
    piy[yn][yi+1] += T
    xi -= 1
    yi += 1
    S8 += 1
  elif T == T9:
    pix[xn-1][xi] += T
    piy[yn+1][yi] += T
    xn -= 1
    yn += 1
    S9 += 1
  elif T == T10:
    pix[xn][xi] += T
    piy[yn+1][yi] += T
    yn += 1
    S10 += 1
  Xn.append(xn)
  Yn.append(yn)
  Xi.append(xi)
  Yi.append(yi)
  print(xn, yn, xi, yi)
  T1,T2,T3,T4,T5,T6,T7,T8,T9, T10 = None, None, None, None, None, None, None, None, None, None
  Tsim += T
print(S1,S2,S3,S4,S5,S6,S7,S8,S9,S10)
TXn = pd.Series(Xn)
TYn = pd.Series(Yn)
TXi = pd.Series(Xi)
TYi = pd.Series(Yi)

fig, axs = plt.subplots(2, 2, figsize=(10,8))
axs[0, 0].plot(TXn.index,TXn.values)
axs[0, 0].axhline(y=sum(Xn)/100000,c='k')
axs[0, 0].set_title('Leeches sanos')
axs[0, 1].plot(TYn.index,TYn.values, 'tab:orange')
axs[0, 1].axhline(y=sum(Yn)/100000,c='k')
axs[0, 1].set_title('Seeds sanos')
axs[1, 0].plot(TXi.index,TXi.values, 'tab:green')
axs[1, 0].axhline(y=sum(Xi)/100000,c='k')
axs[1, 0].set_title('Leeches infectados')
axs[1, 1].plot(TYi.index,TYi.values, 'tab:red')
axs[1, 1].axhline(y=sum(Yi)/100000,c='k')
axs[1, 1].set_title('Seeds infectados')
fig.tight_layout()
print('Leeches sanos:',sum(Xn)/100000)
print('Seeds sanos:',sum(Yn)/100000)
print('Leeches infectados:',sum(Xi)/100000)
print('Seeds infectados:',sum(Yi)/100000)
print('Valores usados: C ',C,'L',L,'M',M,'H',H,'G',G,'N',N,'K',K)
```

Los resultados esperados son algo similar a la figura 6

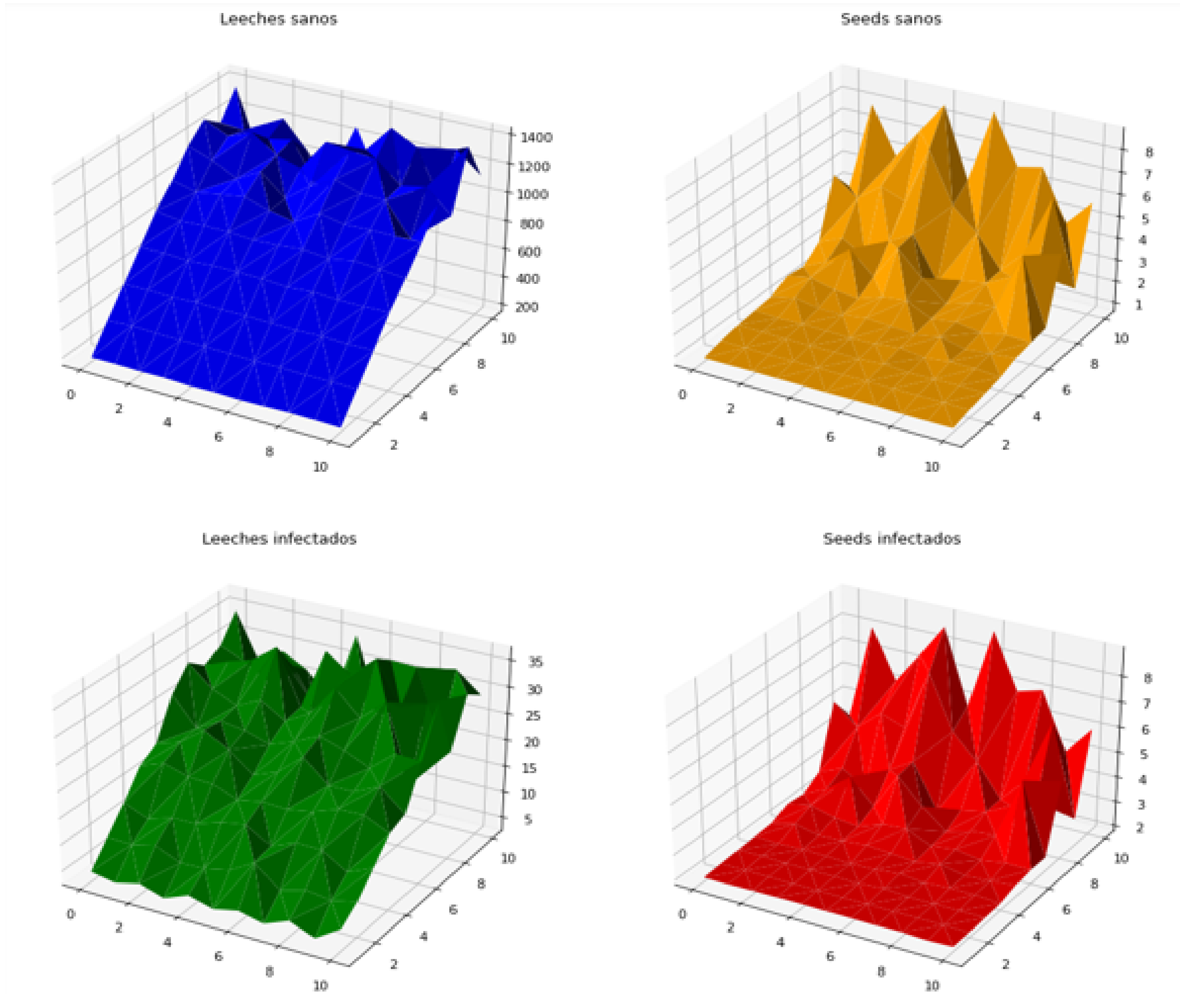

Figura 6: Resultados cadena infectada con contramedidas

Estos parámetros pueden ser variados en más formas, pero no siempre será posible encontrar parámetros que cumplan con el comportamiento esperado.