MODÉLISATION DE LA PROPAGATION D'UNE ÉPIDÉMIE ET OPTIMISATION DE L'IMMUNISATION D'UNE POPULATION

OUAZZANI CHAHDI Mohammed

PLAN

1. Introduction à l'épidémiologie
2. Présentation du modèle SIR
3. Présentation du problème
4. Résolution du problème
5. Bilan

Introduction

- Hippocrate est le premier épidémiologiste (Ve siècle avant J-C).
- L'épidémiologie mathématique voit le jour grâce à Daniel Bernoulli .
- Il développe un modèle pour limiter la propagation du Variole.
- L'OMS annonce l'éradication du Variole en 1980.

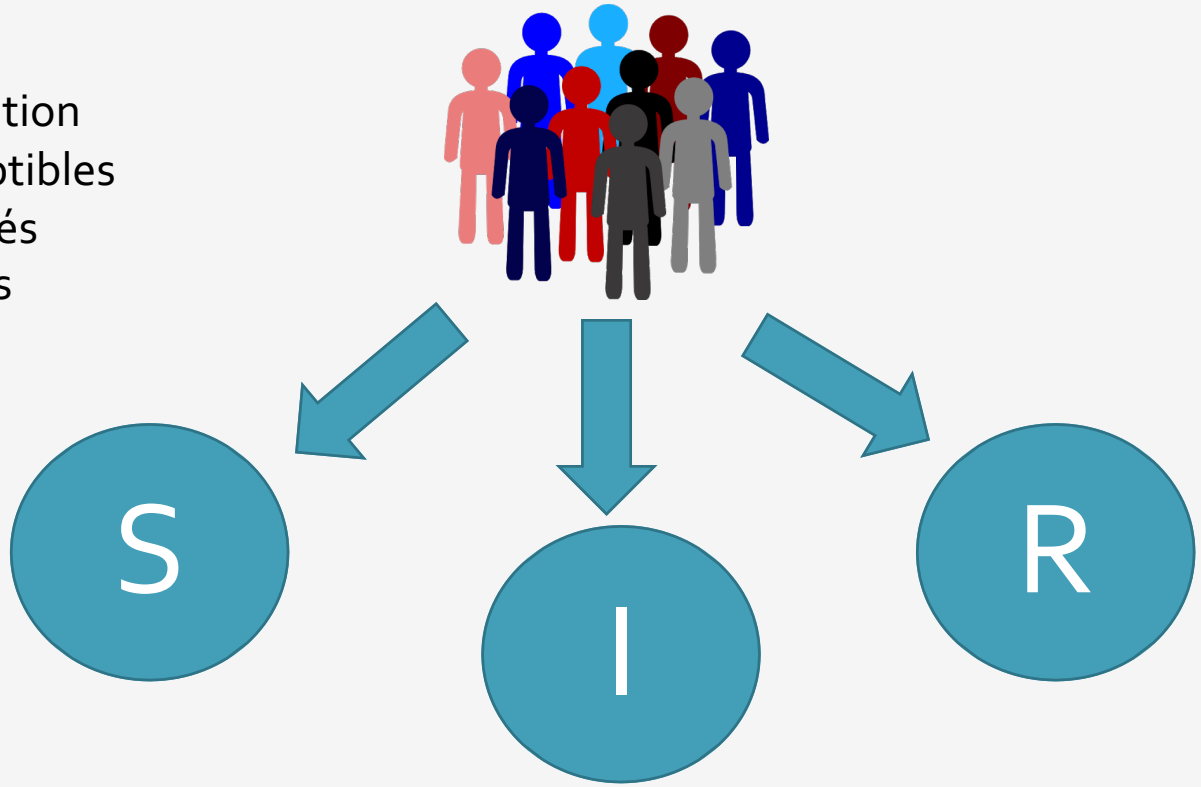
Daniel Bernoulli

(1700- 1782)

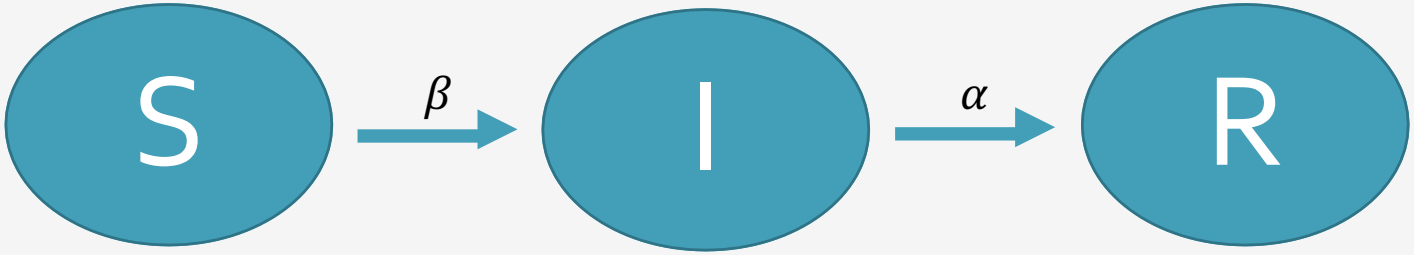
Modèle SIR

$$N = S(t) + R(t) + I(t) = Constant$$

N : Population
S : Susceptibles
I : Infectés
R : Retirés



Modèle SIR



$$\begin{cases} S'(t) = -\beta\, IS & (1) \\ I'(t) = \beta\, IS - \alpha\, I & (2) \\ R'(t) = \alpha\, I & (3) \end{cases}$$

β: Taux de transmission

α: Taux de recouvrement

Modèle SIR

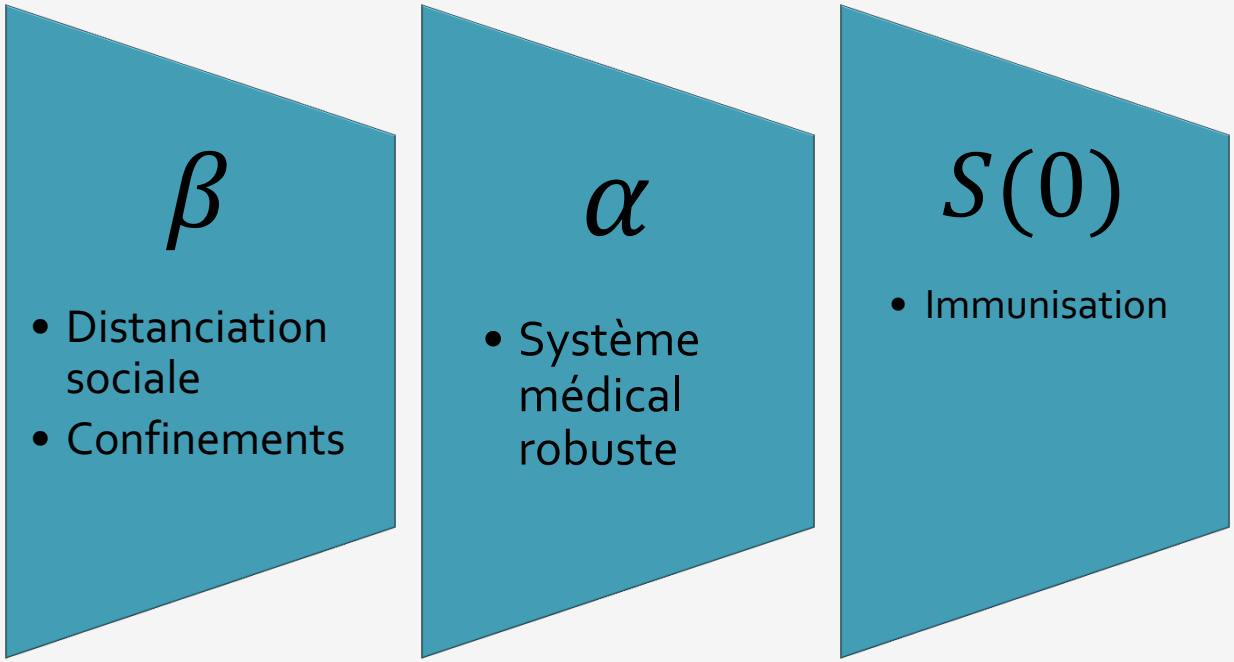
$$I'(t) > 0 \overset{(2)}{\Longrightarrow} \frac{\beta S(t)}{\alpha} > 1$$

Taux de reproduction de base: $$R_0 = \frac{\beta S(0)}{\alpha} \quad (4)$$

Epidémie si : $$R_0 > 1$$

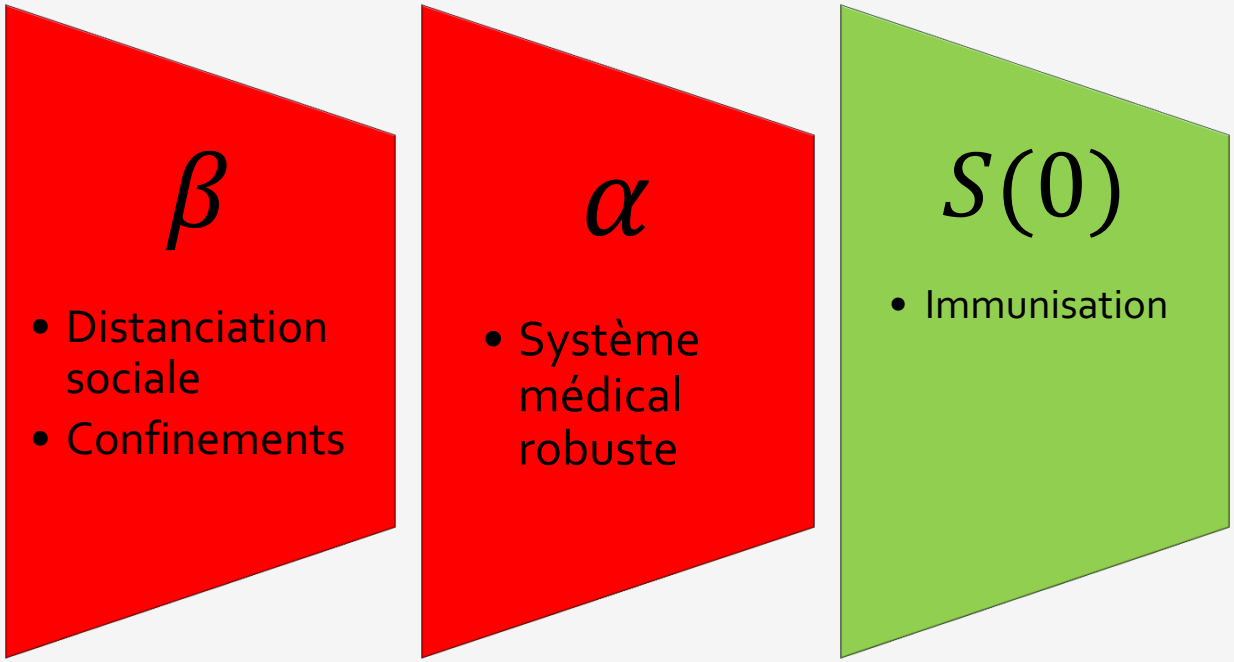
Modèle SIR

$$R_0 = \frac{\beta S(0)}{\alpha} \qquad (4)$$



Modèle SIR

$$R_0 = \frac{\beta S(0)}{\alpha} \qquad (4)$$



Problème

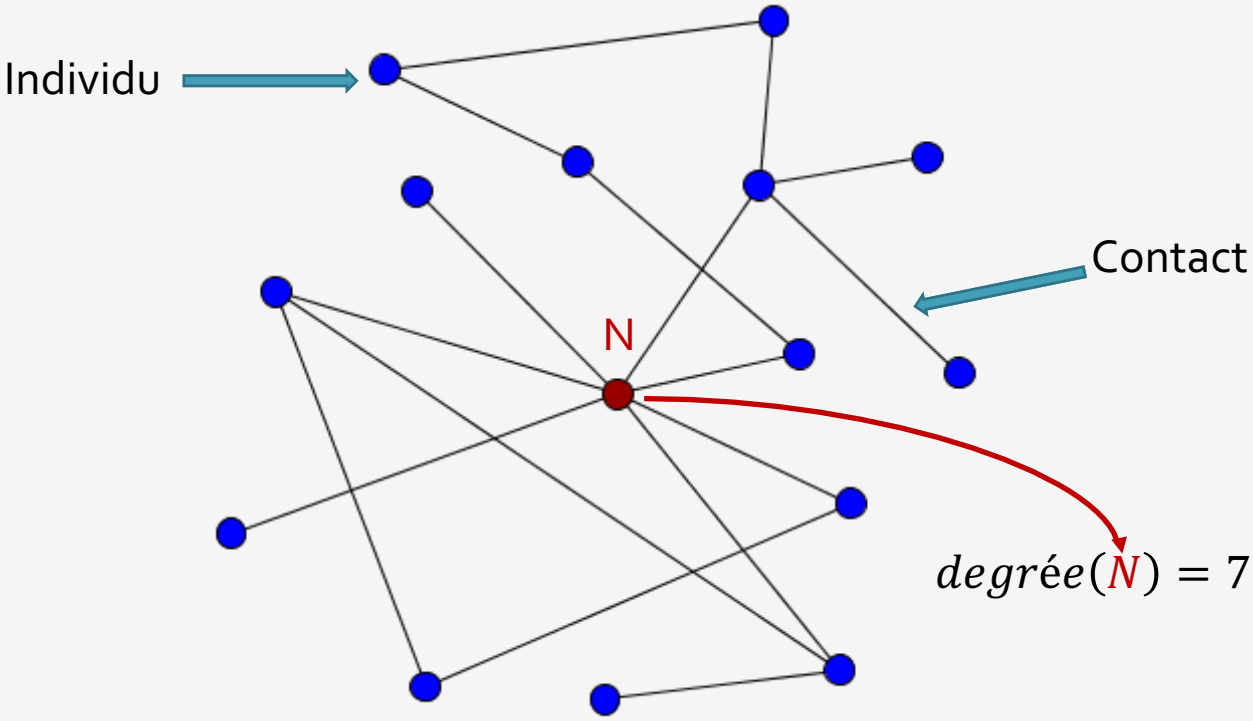
Comment peut-on limiter la propagation d'une épidémie avec le minimum de vaccins ?

Résolution du problème

I. Représenter la population.
II. Modéliser la propagation de l'épidémie dans la population.
III. Introduire 3 différents procédés d'immunisation.
IV. Comparer les différentes approches

Résolution du problème

I. Représenter la population

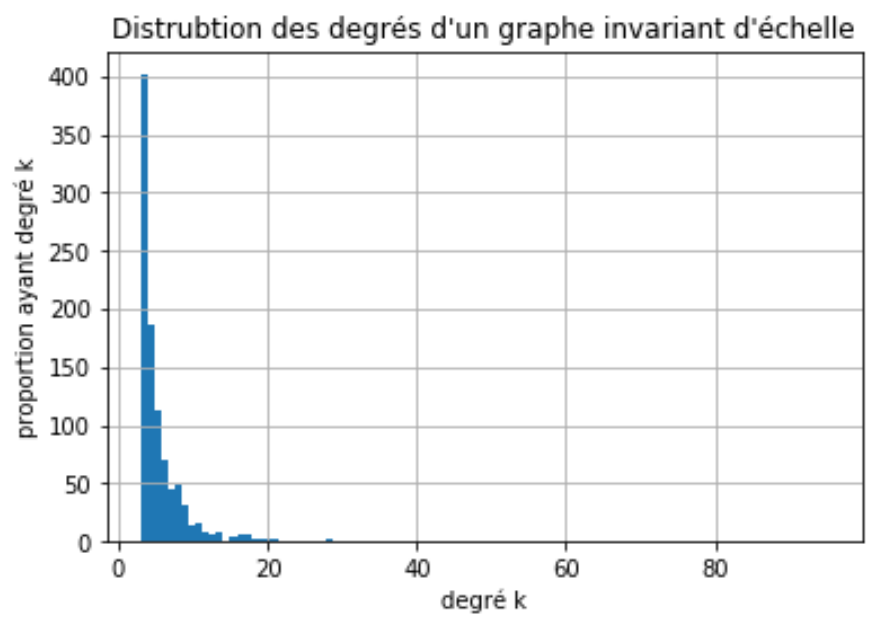


Résolution du problème

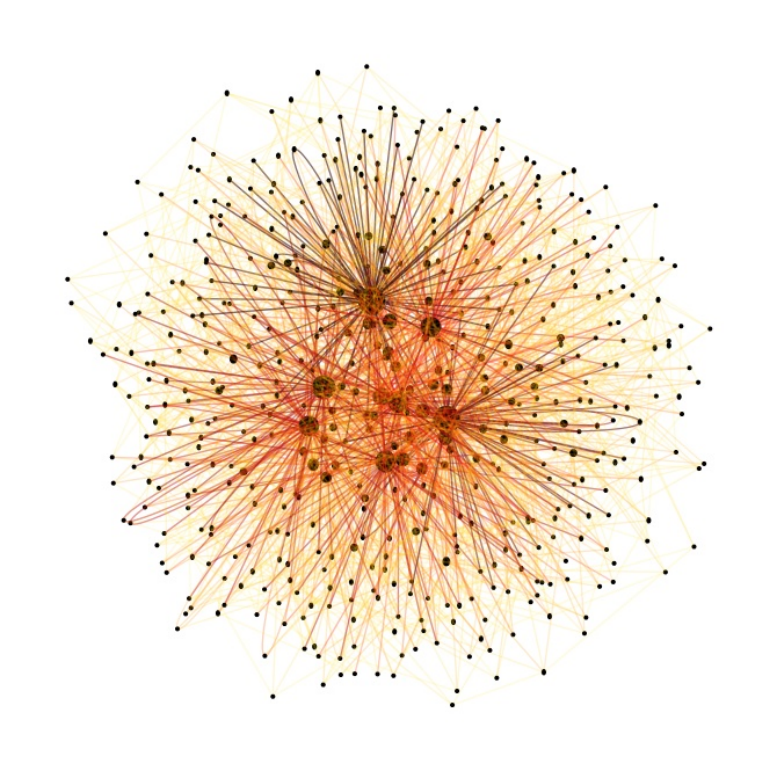
I. Représenter la population

Proportion des nœuds ayant degré k :

$$P(k) \propto \frac{1}{k^{\gamma}}$$



Simulation pour une population N de 1000 individus

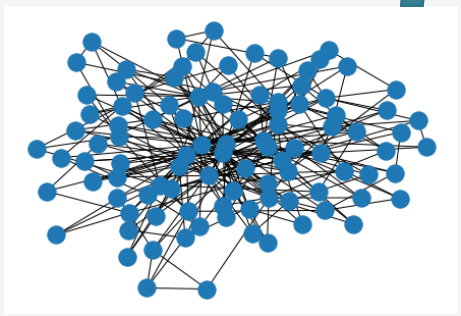


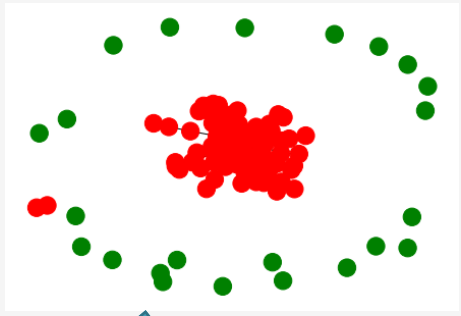
*Helmut G. Katzgraber,Katharina Janzen, and Creighton K. Thomas Boolean decision problems with competing interactions on scale-free networks: Critical thermodynamics
PACS numbers: 75.50.Lk, 75.40.Mg, 05.50.+q, 64.60.-i

Résolution du problème

II. Modéliser la propagation de l'épidémie dans la population.

Extraction du sous-graphe





Connecter les arêtes:

```
for arete in Graphe_initial.edges():
        if random.random()<p_i:
            Sous_graphe.add_edge(*arete)
```

Visualisation pour N = 100 , probabilité d'infection 0.4

Résolution du problème

III. Les procédés d'immunisation

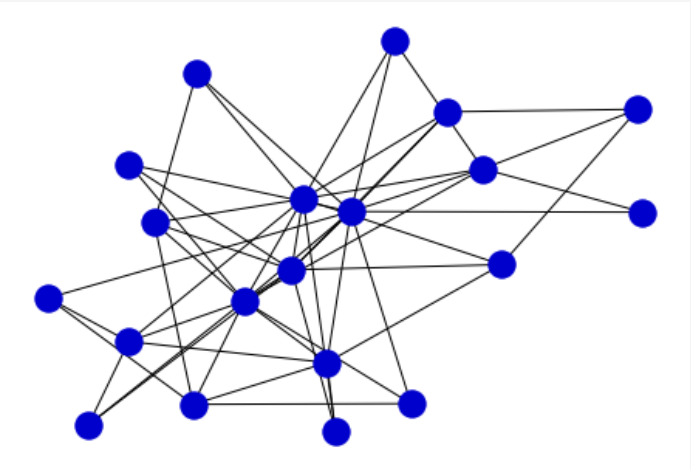
A. Immunisation aléatoire.

B. Immunisation visée.

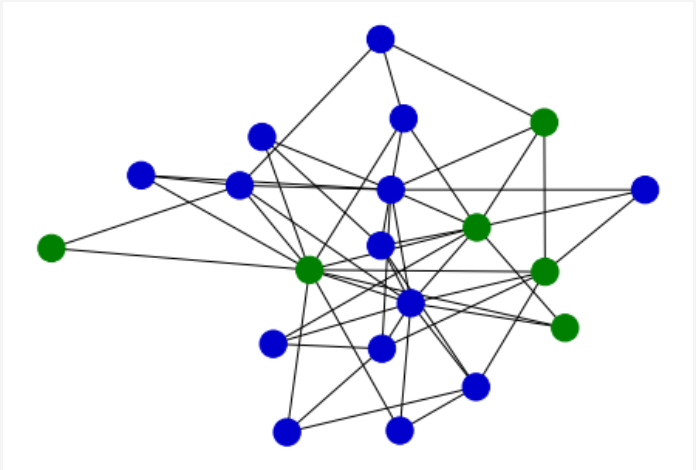
C. Immunisation par connaissances.

Résolution du problème

III. Les procédés d'immunisation
A. Immunisation aléatoire



Immunisation aléatoire de 30% de la population

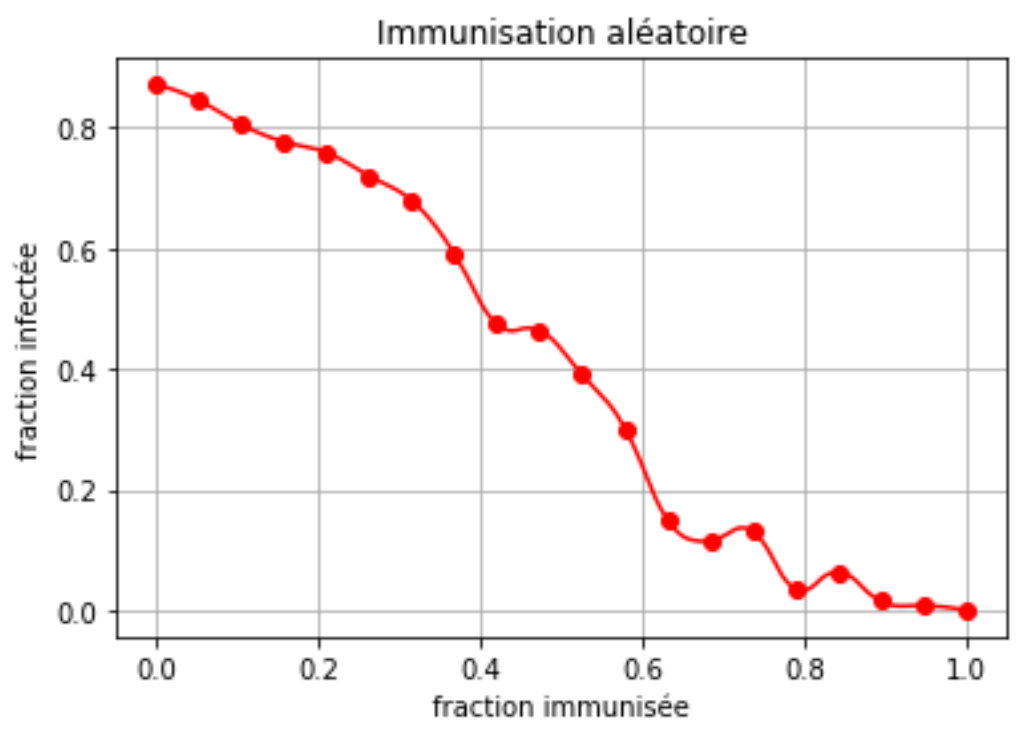


```
A_immuniser = random.sample(list(Graphe_initial.nodes),int(N * f_i))
```

Visualisation pour N = 20 ,
probabilité d'infection 0.6
Fraction immunisé 0.3

Résolution du problème

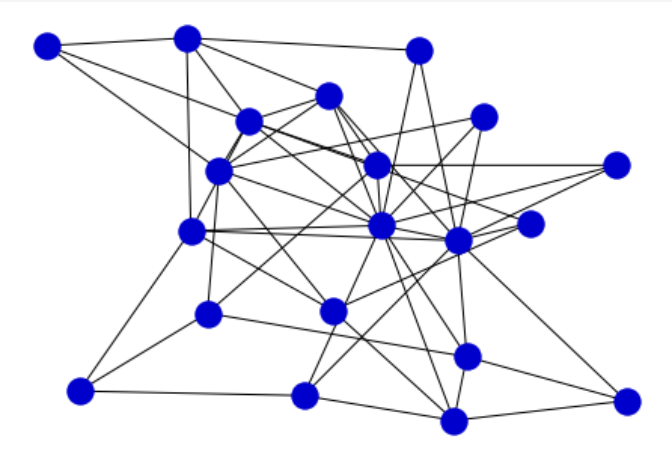
III. Les procédés d'immunisation
A. Immunisation aléatoire



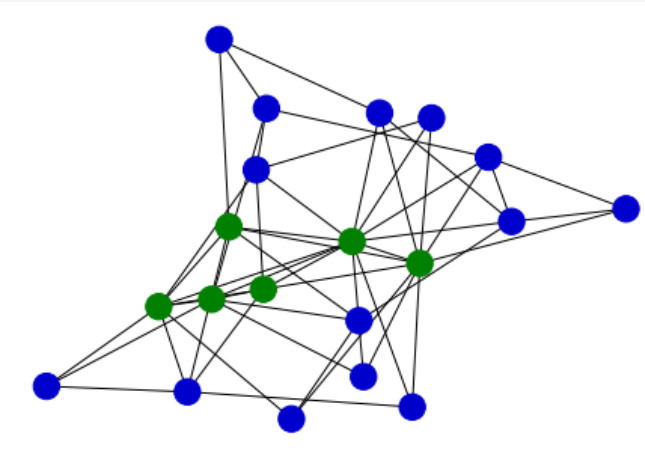
Probabilité d'infection de 0.6
N = 10000
20 simulations

Résolution du problème

III. Les procédés d'immunisation
B. Immunisation visée



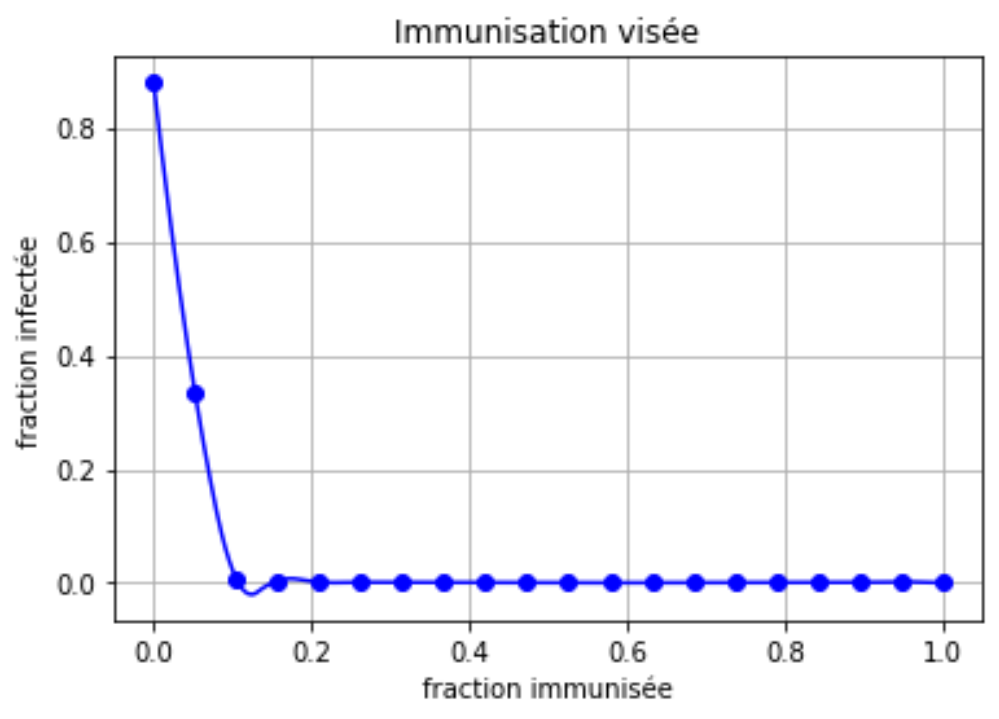
Immunisation visée de 30% de la population



Visualisation pour N = 20 ,
probabilité d'infection 0.6
Fraction immunisé 0.3

Résolution du problème

III. Les procédés d'immunisation
B. Immunisation visée

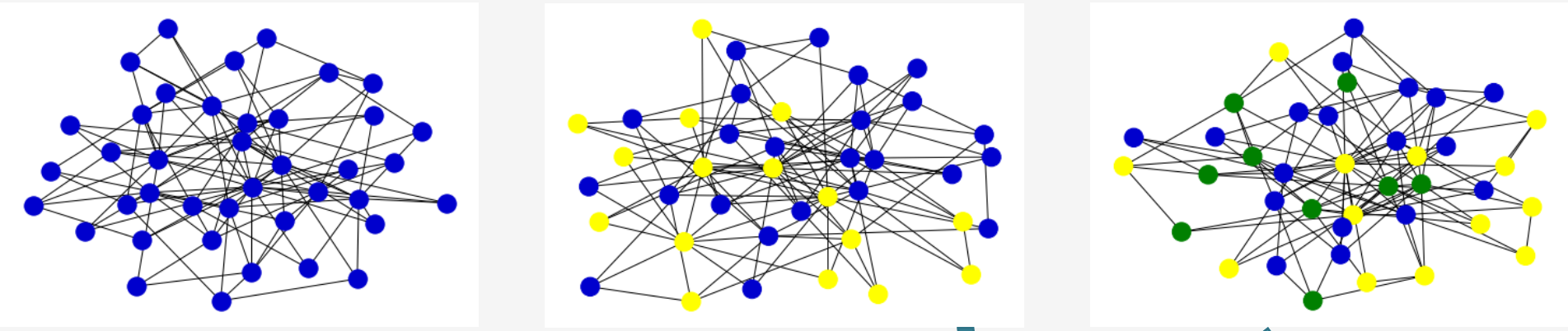


Probabilité d'infection de 0.6
N = 10000
20 simulations

Résolution du problème

III. Les procédés d'immunisation
B. Immunisation par connaissances

Choix d'une fraction aléatoire des nœuds

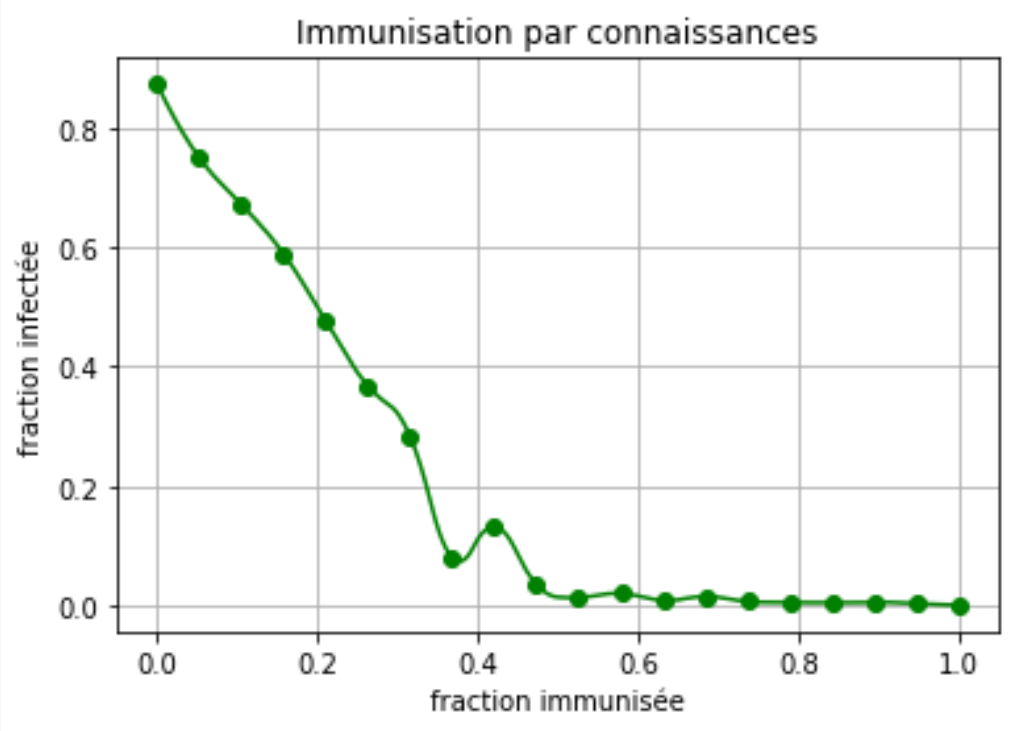


Immuniser une fraction des voisins des nœuds choisis

Visualisation pour N = 50 ,
probabilité d'infection 0.6
Fraction immunisé 0.3

Résolution du problème

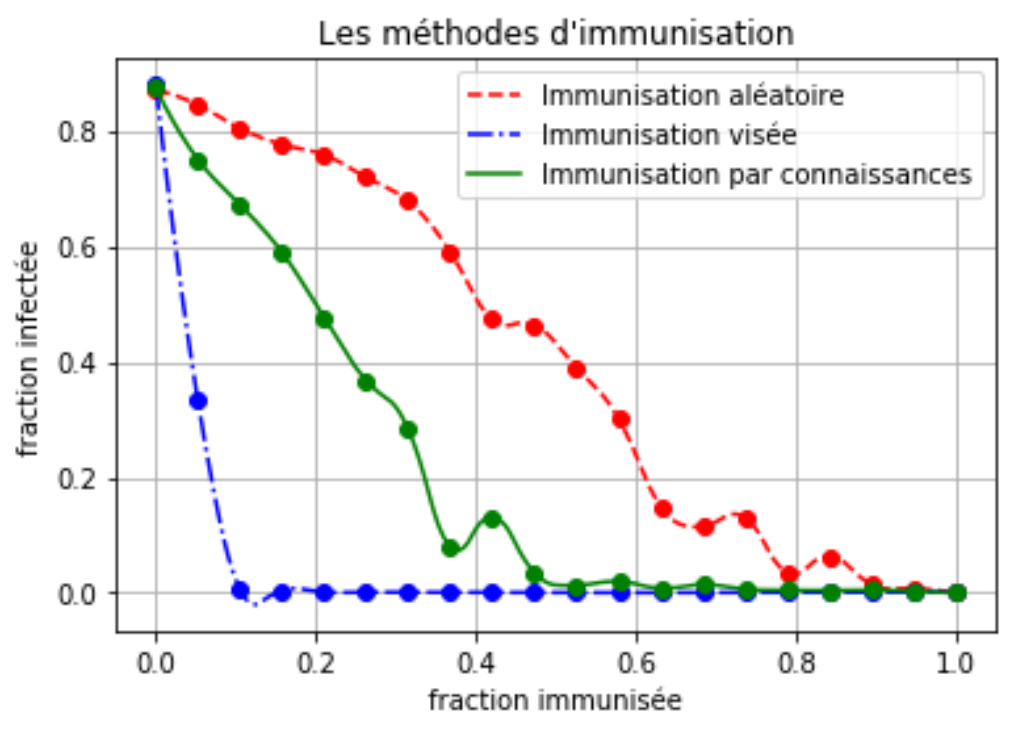
III. Les procédés d'immunisation
B. Immunisation par connaissances



Probabilité d'infection de 0.6
N = 10000
20 simulations

Résolution du problème

III. Comparaison des différentes méthodes



Probabilité d'infection de 0.6
N = 10000
20 simulations

Bilan

Aléatoire

- Nécessite une large proportion immunisé
- N'est pas efficace

Visée

- Très efficace
- Difficile à réaliser

Par connaissances

- Plus efficace que l'immunisation aléatoire
- Plus réalisable que l'immunisation visée

LISTINGS

Algorithme principal de simulation

```python
import numpy as np
import matplotlib.pyplot as plt
import random as rd
import networkx as nx
from scipy import interpolate

#---------------Creation du graphe-----------------------------
N = 10000                               #Population
BA = nx.barabasi_albert_graph(N,2) #Graphe suivant l'algorithme de Barabasi-Albert


p_i = 0.6   #Probabilité d'infection

#-----------------Creation des axes-----------------------------

points = 20    #Nombres de simulations
FA_aleatoire = np.zeros(points) #FA : Fraction Attacké = Fraction infecté
FA_vise = np.zeros(points)
FA_conn = np.zeros(points)
FI = np.linspace(0,1,points) #FI : Fraction immunisée


```

Algorithme principal de simulation

```
23  #---------------Propagation de l'épidemie--------------------
24
25  def percolation_graphe(G,p):
26      H = nx.Graph()   #Creation d'un graphe vide
27      H.add_nodes_from(G.nodes())  #ajouter tous les noeuds de l'ancien graphe
28      for arete in G.edges():    #Pour une certaine proba connecter les noeuds
29          if rd.random()<p:
30              H.add_edge(*arete)
31      return H
32
33  def taille_epidemie(G, p):
34      H = percolation_graphe(G, p)
35      composantes = nx.connected_components(H)
36      taille = 0
37      for c in composantes:
38          if len(c) > taille:
39              taille = len(c)
40      fraction_attacke = float(taille)/G.order()
41
42      return fraction_attacke
43
```

Algorithme principal de simulation

```
44  #------------------Tri du graphe-------------------------
45  def fusion(liste1,liste2):
46      liste=[]
47      i,j=0,0
48      while i<len(liste1)and j<len(liste2):
49          if liste1[i][1]>=liste2[j][1]: #C'est un tri descendant
50              liste.append(liste1[i])
51              i+=1
52          else:
53              liste.append(liste2[j])
54              j+=1
55      if i < len(liste1): liste.extend(liste1[i:])
56      if j < len(liste2): liste.extend(liste2[j:])
57      return liste
58
59
60  def tri_fusion(liste):    #liste des tuples
61      if len(liste)<2:    return liste
62      milieu=len(liste)//2
63      liste1=tri_fusion(liste[:milieu])
64      liste2=tri_fusion(liste[milieu:])
65      return fusion(liste1,liste2)
66
67  def tri_graphe(G):
68      L = list(G.degree)
69      return tri_fusion(L)
70
```

Algorithme principal de simulation

```python
#------------------Les fonctions d'immunisation------------------

def immunisation_aleatoire(G,f_i):
    G = G.copy()
    A_immuniser = rd.sample(list(G.nodes),int(N * f_i))
    G.remove_nodes_from(A_immuniser)
    return taille_epidemie(G,p_i)

def immunisation_vise(G,f_i):
    G = G.copy()
    G_trie = tri_graphe(G)  #Liste triée de la forme [(noeud, degree)] (tri)
    A_immuniser = []
    for i in range(int(f_i*N)): A_immuniser.append(G_trie[i][0])
    G.remove_nodes_from(A_immuniser)
    return taille_epidemie(G,p_i)

def immunisation_par_connaissance(G,f_i):
    G = G.copy()
    p = rd.random()
    choisi = rd.sample(list(G.nodes),int(p*N))
    N_immunise = int(N*f_i)
    A_immuniser = []
    while N_immunise > 1:
        for noeud in choisi:
            voisins = list(G.neighbors(noeud))
            nv = len(voisins)
            H = rd.randint(0,N_immunise)
            if nv >= H:
                L = rd.sample(voisins, H)
                N_immunise = N_immunise - H
            else:
                L = rd.sample(voisins, nv)
                N_immunise = N_immunise - nv
            A_immuniser.extend(L)
    G.remove_nodes_from(A_immuniser)
    return taille_epidemie(G,p_i)

```

Algorithme principal de simulation

```
108  #-----------------Simulation de la propagation----------------------
109  for i in range(points-1):
110      f_i = FI[i]
111      FA_aleatoire[i] = immunisation_aleatoire(BA,f_i)
112      FA_vise[i] = immunisation_vise(BA,f_i)
113      FA_conn[i] = immunisation_par_connaissance(BA,f_i)
114
115  #----------------------Interpolation-------------------------------
116
117  x = FI
118  y1,y2,y3 = FA_aleatoire, FA_vise, FA_conn
119  f1 = interpolate.interp1d(x,y1,kind = "quadratic")
120  f2 = interpolate.interp1d(x,y2,kind = "quadratic")
121  f3 = interpolate.interp1d(x,y3,kind = "quadratic")
122  xn = np.linspace(0,1,1000)
123  yn1,yn2,yn3 = f1(xn),f2(xn),f3(xn)
124
```

Algorithme principal de simulation

```
125  #-------------------------Traçage-----------------------------------
126
127  plt.plot(x,y1,'or')
128  plt.plot(xn,yn1,'--r', label = "Immunisation aléatoire")
129  plt.plot(x,y2,'ob')
130  plt.plot(xn,yn2,'-.b', label ="Immunisation visée" )
131  plt.plot(x,y3,'og')
132  plt.plot(xn,yn3,'-g', label = "Immunisation par connaissances" )
133  plt.xlabel("fraction immunisée")
134  plt.ylabel("fraction infectée")
135  plt.title("Les méthodes d'immunisation")
136  plt.legend()
137  plt.grid()
138  plt.show()
```

Les algorithmes de visualisation: Distribution des degrés(Page12)

```python
import networkx as nx
import matplotlib.pyplot as plt

G = nx.barabasi_albert_graph(1000,2)

degrees = []
L = list(G.degree)
for e in L: degrees.append(e[1])

#Traçage du histogramme
plt.hist(degrees, bins = 100)
plt.grid()
plt.xlabel("degré k")
plt.ylabel("proportion ayant degré k")
plt.title("Distrubtion des degrés d'un graphe invariant d'échelle")

```

Les algorithmes de visualisation: Propagation de l'épidémie(Page13)

```python
import networkx as nx
import random as rd
import matplotlib.pyplot as plt

N = 100
p_i = 0.4

G = nx.barabasi_albert_graph(N,2)
H = nx.Graph()

plt.figure(1)
nx.draw(G)

H.add_nodes_from(G.nodes())  #Sous graphe de G contenant les noeuds
plt.figure(2)
nx.draw(H, node_color = "green")

for arete in G.edges(): #Propagation: les arcs se forment pour proba < p_i
        if rd.random()<p_i: H.add_edge(*arete)

```

Les algorithmes de visualisation: Propagation de l'épidémie(Page13)

```
21  #Coloriage
22  color_map = []
23  nodes = list(H.nodes())
24  for comp in nx.connected_components(H):
25      if len(comp) <= 1:
26          a = list(comp)[0]
27          nodes.remove(a)
28  for n in H:
29      if n in nodes:
30          color_map.append("red")
31      else:
32          color_map.append("green")
33
34  plt.figure(3)
35  nx.draw(H, node_color = color_map)
36
```

Les algorithmes de visualisation: Immunisation aléatoire (Page15)

```python
import networkx as nx
import random as rd
import matplotlib.pyplot as plt

N = 20
p_i = 0.6
f_i = 0.3

G = nx.barabasi_albert_graph(N,2)
plt.figure(1)
nx.draw(G, node_color = "mediumblue")
A_immuniser = random.sample(list(Graphe_initial.nodes),int(N * f_i))
color_map =[]
for noeud in G:
    if noeud in A_immuniser:
        color_map.append("green")
    else:
        color_map.append('mediumblue')
plt.figure(2)
nx.draw(G,node_color = color_map)

```

Les algorithmes de visualisation: Immunisation visée (Page17)

```python
import networkx as nx
import random as rd
import matplotlib.pyplot as plt

N = 20
p_i = 0.6
f_i = 0.3

G = nx.barabasi_albert_graph(N,2)
plt.figure(1)
nx.draw(G, node_color = "mediumblue")

def fusion(liste1,liste2):
def tri_fusion(liste):
def tri_graphe(G):

G_trie = tri_graphe(G)  #Liste triée de la forme [(noeud, degree)] (tri)
A_immuniser = []
for i in range(int(f_i*N)): A_immuniser.append(G_trie[i][0])

color_map =[]
for noeud in G:
    if noeud in A_immuniser:
        color_map.append("green")
    else:
        color_map.append('mediumblue')
plt.figure(2)
nx.draw(G,node_color = color_map)
```

Les algorithmes de visualisation: Immunisation par connaissances (Page19)

```python
import networkx as nx
import random as rd
import matplotlib.pyplot as plt

N = 40
p_i = 0.6
f_i = 0.3

G = nx.barabasi_albert_graph(N,2)
plt.figure(1)
nx.draw(G, node_color = "mediumblue")


p = rd.random()
choisi = rd.sample(list(G.nodes),int(p*N))


color_map1 = []
for noeud in G:
    if noeud in choisi:
        color_map1.append("yellow")
    else:
        color_map1.append('mediumblue')
plt.figure(2)
nx.draw(G,node_color = color_map1)

```

Les algorithmes de visualisation: Immunisation par connaissances (Page19)

```
N_immunise = int(N*f_i)
A_immuniser = []
while N_immunise > 1:
    for noeud in choisi:
        voisins = list(G.neighbors(noeud))
        nv = len(voisins)
        H = rd.randint(0,N_immunise)
        if nv >= H:
            L = rd.sample(voisins, H)
            N_immunise = N_immunise - H
        else:
            L = rd.sample(voisins, nv)
            N_immunise = N_immunise - nv
        A_immuniser.extend(L)


color_map2 =[]
for noeud in G:
    if noeud in A_immuniser:
        color_map2.append("green")
    elif noeud in choisi:
        color_map2.append("yellow")
    else:
        color_map2.append('mediumblue')
plt.figure(3)
nx.draw(G,node_color = color_map2)
```