Percolation et propagation des feux de forêts

Madeleine Hueber

Année 2020-2021

Table des matières

1 Introduction

1.1 Position du problème

Ces dernières années nous ont montrées à quel point les feux de forêts peuvent être destructeurs. Ainsi contrôler la propagation de ces feux apparaît comme un enjeu environnemental majeur. Pour pouvoir les maîtriser il est nécessaire d'être capable de prévoir l'évolution d'un incendie. Le modèle mathématique de percolation permet ainsi de prévoir l'évolution d'un feu au sein d'une forêt et l'ampleur des dégâts potentiels. Les questions qui nous intéressent sont donc les suivantes : À quelle condition un feu ravage-t-il une forêt entière, comment le modèle de percolation est-il adapté à cette étude et quels résultats fournit-il ?

1.2 Introduction du modèle

Construire un graphe de percolation :

Pour construire un tel graphe on se donne un réseau régulier infini, par exemple $\mathbb{Z}^2$ et on attribue à chaque arrête une probabilité p d'exister : c'est le modèle de percolation de liens. On peut également choisir d'attribuer à chaque sommet une probabilité p :c'est le modèle de percolation de site (Figure 1). On construit ainsi $(X_{(u,v)})_{<u,v>\in E}$ des variables aléatoires i.i.d sur $\{0,1\}$ telles que $X_{(u,v)} \sim B(p)$. On construit de cette manière un graph G_p associé à la probabilité p.
Il s'agit ensuite d'étudier l'existence de composantes connexes infinies. En effet cette existence traduit le fait que le feu pourrait se propager dans toute la forêt.

2 Simulations

Pour mieux se rendre compte du phénomène il est possible de simuler la propagation d'un feu au sein d'une forêt grâce à un algorithme Python. Pour cela on créer une matrice de taille nxn dont les coefficients prennent la valeur 1 (présence d'un arbre) avec une probabilité p et 0 (emplacement vide) avec une probabilité 1-p. Une fois la forêt crée on peut y faire se propager un feu grâce à un algorithme fonctionnant de manière itérative. On observe alors deux cas de figure selon la valeur du paramètre p : Pour une probabilité faible le feu se propage peu ; au contraire pour une valeur de p assez élevée le feu détruit presque entièrement la forêt.(Figure 2) La question qui se pose est alors de déterminer à partir de quelle valeur de p la forêt brûle entièrement, c'est à dire qu'il y a une composante connexe infinie. Notons que le cadre qui nous intéresse est le réseau infini, pourtant l'algorithme Python nous restreint à des forêts de taille finie. Ainsi pour pallier à ce problème il est nécessaire de prendre des valeurs de n élevées pour se rapprocher au mieux du réseau infini ($n \approx 200$). On considère alors qu'il existe une composante connexe infinie quand, en démarrant du haut de la forêt le feu se propage jusqu'en bas. On peut ainsi créer un algorithme qui pour une forêt donnée détermine si il existe une composante connexe infinie et finalement tracer la courbe donnant la fraction de forêt brûlées en fonction de p.(Figure 3) On remarque alors l'existence d'une probabilité de seuil pour p=p_c ≈0.59 :Si p< p_cl'existence d'une composante connexe infinie est un évènement négligeable tandis que pour p> p_c cette existence est un évènement presque sur.

3 Preuve de l'existence d'une probabilité de seuil

Il est alors possible de prouver l'existence de cette probabilité de seuil. Pour cela on introduit la fonction $\theta(p)$:

Définition 1 $\theta(p) = P(|C(Gp,0)| = \infty)$ *où C(Gp,0) représente la composante du graphe Gp contenant l'origine.*

On a :
$\theta(0) = 0$ $\theta(1) = 1$
θ est croissante

Définition 2 $p_c = sup\{p \in [0,1], \theta(p) = 0\}$

Théoreme 1 *(loi du zéro-un de Kolmogorov) Tout événement dont la réalisation dépend d'une suite de variables aléatoires indépendantes mais ne dépend d'aucun sous-ensemble fini de ces variables est de probabilité 0 ou 1.*

Ici l'événement est : A :"Il existe une composante connexe infinie"
On a alors $P(A) \geq \theta(p)$ donc si p> p_c $P(A) > 0 \Rightarrow P(A) = 1$
et si p<p_c

$$P(A) \leq \sum_{u \in \mathbb{Z}^2} P(|C(Gp,u)| = \infty) = \sum_{u \in \mathbb{Z}^2} P(|C(Gp,0)| = \infty) \quad (1)$$

$$= \sum_{u \in \mathbb{Z}^2} \theta(p) = 0 \quad (2)$$

4 Calculs de pc

4.1 Réseau en dimension 1

Pour le réseau en dimension 1 (Figure 4) on peut calculer facilement la probabilité critique :
La proba $P_s(p)$ qu'un site donné appartienne à un cluster de taille s est :

$P_s(p) = sp^s(1-p)^2$

La probabilité d'appartenir à un cluster infini P_∞ est alors donnée par :
$p = \sum_{s=1}^{+\infty} P_s(p) + P_\infty(p)$

Si $p < 1$: $P_\infty(p) = p - p(1-p)^2 \sum_{s=1}^{+\infty} sp^{s-1} = 0$
Si p=1 : $P_\infty(p) = 1$ Donc $p_c = 1$

Dans le cas p<1 on peut égalment déterminer la taille moyenne $S(p)$ d'un cluster :

$S(p) = \sum_{s=1}^{+\infty} s \times \frac{P_s(p)}{p} = (1-p)^2 \sum_{s=1}^{+\infty} s^2 p^{s-1} = \frac{1+p}{1-p}$

4.2 Réseau de Bethe

Définition 3 *Le Réseau de Bethe est un réseau sans cycle de sites équivalents ayants tous z voisins. (Figure 5)*

Pour le réseau de Bethe on a $p_c = \frac{1}{z-1}$
On peut également déterminer la taille moyenne d'un cluster si $p < p_c$:
$S(p) = 1 + zT$ avec T la taille moyenne d'une branche donnée par :

$T = (1-p) \times 0 + p \times (1 + (z-1)T) = \frac{p}{1-p(z-1)} = \frac{p_c p}{p_c - p}$

5 Résultats sur pc

Dans le cadre du réseau donné par $\mathbb{Z}^2$ on a un encadrement de p_c.

5.1 Preuve de $p_c \geq \frac{1}{3}$

Pour prouver cette première inégalité on utilise les chemins issus de l'origine :

Définition 4 *$\sigma(n)$ =nb de chemins sans boucles de longueur n et d'origine (0,0)*

Pour construire un tel chemin on dispose de 4 choix pour la première arêtes et 3 choix pour les suivantes (car il est impossible de revenir en arrière). (Figure 6) On obtient donc $\sigma(n) \leq 4 \times 3^{n-1}$.

De plus on a l'équivalence : la composante connexe contenant l'origine est infinie $\Leftrightarrow$ pour tout entier n il existe un chemin dans Gp de longueur n partant de l'origine.
Ainsi : $\{C(Gp, 0) = \infty\} = \bigcap_{n=1}^{+\infty} \{N(n, Gp) \geq 1\}$
avec $\{N(n, Gp)\}$= nombre de chemins de longueur n dans Gp partant de 0
D'où

$$\theta(p) = P(\bigcap_{n=1}^{+\infty} \{N(n, Gp) \geq 1\}) \tag{3}$$
$$\leq P(\{N(n, Gp) \geq 1\}) \ \forall n \in \mathbb{N} \tag{4}$$
$$= P(\bigcup_{i=1}^{\sigma(n)} \{Y_i(Gp) = 1\}) \tag{5}$$

Définition 5 *$Y_i(Gp) = 1$ si le ième chemin de longueur n est dans Gp, 0 sinon*

d'où $P(Y_i(Gp) = 1) = p^n$

$\theta(p) \leq 4 \times 3^{n-1} \times p^n = 4p \times (3p)^{n-1} \ \forall n \in \mathbb{N}$

Donc si $p < \frac{1}{3}$ on obtient $\theta(p) = 0$ en faisant tendre n vers l'infini.

5.2 Preuve de $p_c \leq \frac{2}{3}$

Définition 6 *On définit le dual Gp' d'un graph Gp en se plaçant sur la grille duale : chaque arrête existe si elle ne croise aucune arrêtes de Gp.(Figure 7)*

On a alors $|C(Gp,0)| < \infty \Leftrightarrow$ Il existe un circuit de Gp' entourant 0

Définition 7 *$\rho(n)$ = nb de circuits de taille n entourant (0,0).*

Pour construire un tel circuit on choisit un premier point (origine du circuit) de la forme $(\frac{1}{2}, \frac{1}{2}+i), i \in \{0,...,n-1\}$:On a n choix, puis on choisit un chemin de longueur n-1.

D'où $\rho \leq n\sigma(n-1)$

Définition 8 *Pour tout entier naturel k on note B_k l'ensemble des points compris dans le carré de centre (0,0) et de coté k. Et C_k l'ensemble des circuits qui entourent B_k (Figure 8)*

On a l'équivalence suivante : Aucun circuit de C_k n'est circuit de Gp' $\Leftrightarrow$ Il existe au moins un sommet u de Δ_k tel que $|C(Gp,u)| = \infty$

$$P(\bigcup_{u\in\Delta_k}\{|C(Gp,u)|=\infty\}) = P(\bigcap_{c\in C_k}\{c\,pas\,circuit\,de\,Gp'\}) \tag{6}$$

$$= 1 - P(\bigcup_{c\in C_k}\{c\,circuit\,de\,Gp'\}) \tag{7}$$

$$\geq 1 - \sum_{c\in C_k} P(\{c\,circuit\,de\,Gp'\}) \tag{8}$$

Pour un circuit c de longueur n :
$P(\{$c circuit de Gp'$\}) = (1-p)^n$ et $\forall c \in C_k$ c est de longueur $\geq 4k$

$$\sum_{c\in C_k} P(\{c\,circuit\,de\,Gp'\}) \leq \sum_{n=4k}^{+\infty} n\sigma(n-1)\times(1-p)^n \leq \frac{4}{9}\sum_{n=4k}^{+\infty}(3(1-p))^n n \tag{9}$$

Donc si p$> \frac{2}{3}$ $\sum(3(1-p))^n$ CV

Ainsi pour k assez grand :

$$\frac{4}{9}\sum_{n=4k}^{+\infty}(3(1-p))^n n < 1 \Rightarrow P(\bigcup_{u\in\Delta_k}\{|C(Gp,u)|=\infty\}) > 0 \tag{10}$$

$$\Rightarrow P(\{|C(Gp,0)|=\infty\}) > 0 \tag{11}$$

Donc $p_c \leq \frac{2}{3}$

6 Références bibliographiques

[1] Jarry M. et Campet P. : Modélisation d'un feu de forêt :
http ://www.mathom.fr/mathom/sauvageot/Modelisation/Graphes/Feux_foret.pdf

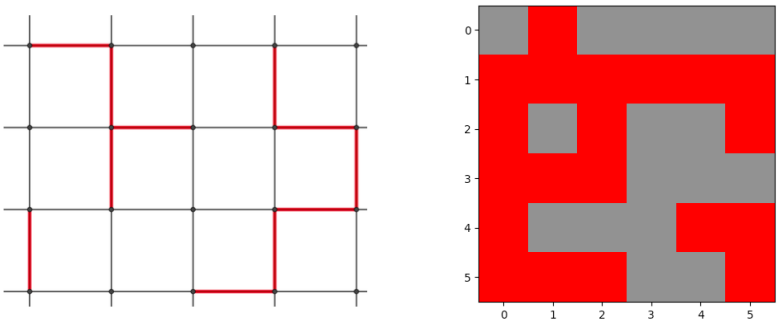
[2] Brémaud P. : Discrete Probability Models and Methods : Springer, 2017

[3] Christensen, K. : Christensen, K. : Percolation theory :
https ://web.mit.edu/ceder/publications/Percolation.pdf

[4] Audibert P. : Simulation d'un feu de forêt :
http ://www.pierreaudibert.fr/tra/feudeforet.pdf

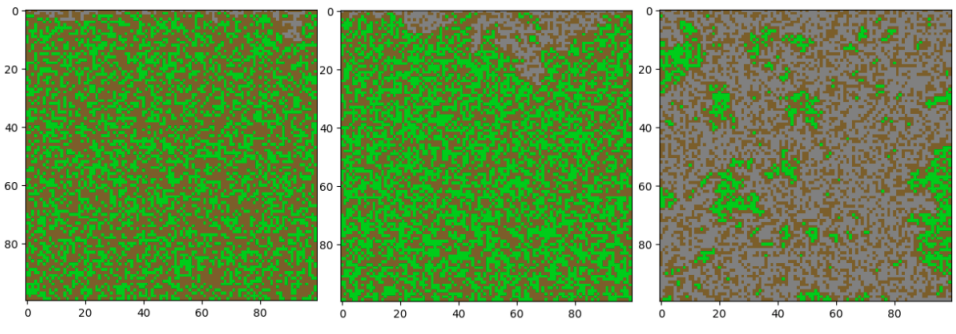
7 Annexes

Figure 1 :



Percolation de liens Percolation de sites

Figure 2 :



n=100 p=0,4 n=100 p=0,5 n=100 p=0,6

Figure 3 :

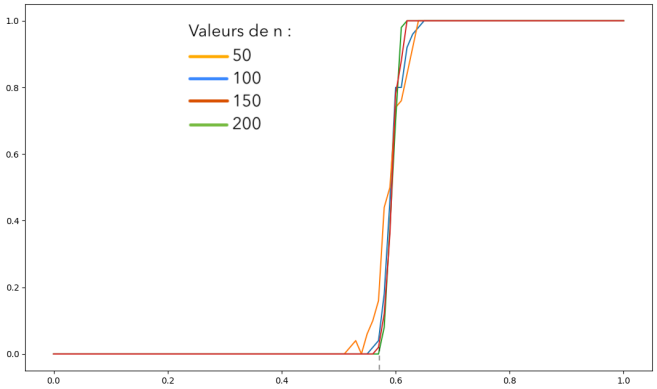
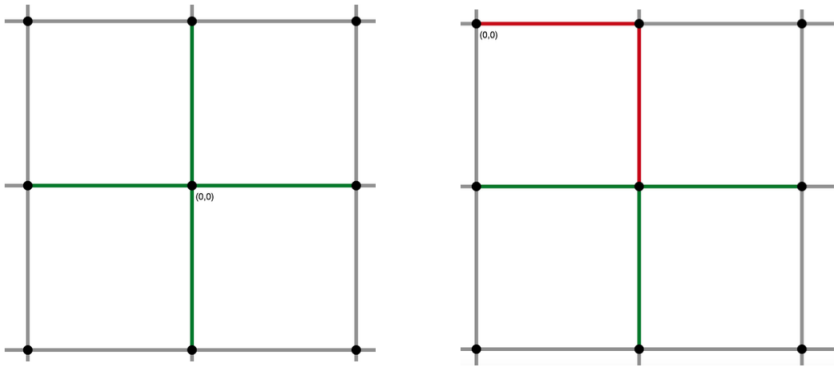


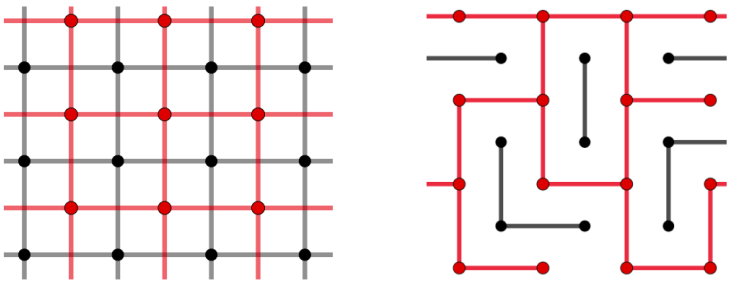
Figure 4 :



1ere arrête : 4 choix

n-1 autres arrêtes : 3 choix

Figure 5 :



Grille duale (en rouge)

Graphe duale Gp' (en rouge)
d'un graph Gp (en noir)

Figure 6 :

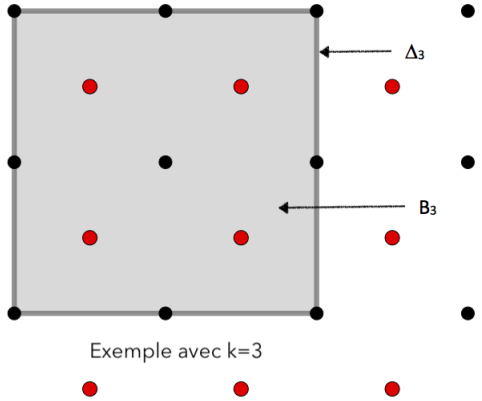
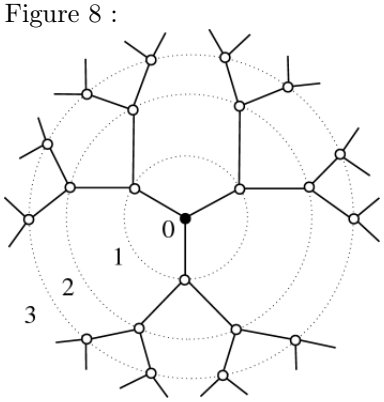


Figure 7 :

Figure 8 :



Codes Python :

```
# TIPE_percolation.py

01| ##TIPE 1: definir un graph= forêt finis mais avec un grd nb
de cases 1 case = 1 arbre H=Graph(n,p)
02|
03| import numpy as np
04| import random
05| from matplotlib import pyplot as plt
06| from matplotlib import colors as c
07|
08| def Graph(n,p):
09|     G= np.zeros((n,n))
10|     for i in range (n):
11|         for j in range (n):
12|             if random.random()<=p:
13|                 G[i][j]=1
14|     return G
15| ## ici on ne peux travailler que sur des graphs finis pour
déterminer si l y a ou non percolation on travaille sur des n
assez grands (n=100 par ex) et on cherche si il existe un chemin
reliant le haut du graph au bas si oui : il y a percolation
sinon non c'est le but de la fonction percolation / perco donne
le résultat Vrai ou Faux la fct percolation elle donne le graph
après incendie
16|
17| def percolation(H):
18|     G=np.copy(H)
19|     Chemin=[]
20|     n=len(G)
21|     for j in range (n):
22|         if G[0][j]==1:
23|             Chemin.append((0,j))
24|             G[0][j]=2
25|     while len(Chemin)>0:
26|         (i,j)=Chemin.pop()
27|         if j<n-1 and G[i][j+1]==1:
28|             Chemin.append((i,j+1))
29|             G[i][j+1]=2
30|         if j>0 and G[i][j-1]==1:
31|             Chemin.append((i,j-1))
32|             G[i][j-1]=2
33|         if i<n-1 and G[i+1][j]==1:
34|             Chemin.append((i+1,j))
35|             G[i+1][j]=2
36|         if i>0 and G[i-1][j]==1:
37|             Chemin.append((i-1,j))
38|             G[i-1][j]=2
39|     return G
40|
41| def perco(H):
```

1

```
    G=percolation(H)
    n=len(G)
    for j in range(n):
        if G[n-1][j]==2:
            return True
    return False

## pour afficher :
plt.imshow(percolation(H),cmap=c.ListedColormap(['#785E2F','#31B
F1B','#DC1E1E'])) puis plt.show() /

def proba(n,p,m):
    nb=0
    for i in range (m):
        G=Graph(n,p)
        if perco(G):
            nb+=1
    return nb/m

X=[0.01*k for k in range (101)]
Y1=[proba(100,x,50) for x in X]
Y2=[proba(50,x,50) for x in X]
Y3=[proba(200,x,50) for x in X]
Y4=[proba(150,x,50) for x in X]

## pour afficher plt.plot(X,Y) plt.show()

## feu de forêt avec choix de l'origine (x,y)

def feux (n,p,x,y):
    G=Graph(n,p)
    A=[(x,y)]# A compte les arbres entrains de bruler
    while len(A)>0:#tant que des arbres brules
        for (i,j) in A:
            if j<n-1 and G[i][j+1]==1:
                A.append((i,j+1))
                G[i][j+1]=2
            if j>0 and G[i][j-1]==1:
                A.append((i,j-1))
                G[i][j-1]=2
            if i<n-1 and G[i+1][j]==1:
                A.append((i+1,j))
                G[i+1][j]=2
            if i>0 and G[i-1][j]==1:
                A.append((i-1,j))
                G[i-1][j]=2
            A.remove((i,j))
    return G
```

2

```
# animation_forêt.py

##ANIMATION
import tkinter
from tkinter import Canvas, Tk
COLORS=["#785E2F", "lime green", "red", "grey"]

def Graph(n,p):
    G= np.zeros((n,n))
    for i in range (n):
        for j in range (n):
            if random.random()<=p:
                G[i][j]=1
    return G
def voisins(n,i,j):
    return[(a,b) for (a,b) in [(i,j-1),(i,j+1),(i-1,j),
(i+1,j)]if a in range(n) and b in range(n)]

def updateforet(F):
    n=len(F)
    A=[] #les arbres qui vont prendre feux
    for x in range(n):
        for y in range (n):
            if F[x][y]==2:
                F[x][y]=3
                for (i,j) in voisins(n,x,y):
                    if F[i][j]==1:
                        A.append((i,j))
    for (i,j) in A:
        F[i][j]=2
def cellule(F,x,y):
    A=(unit*x,unit*y)
    B=(unit*(x+1),unit*(y+1))
    etat=F[x][y]
    couleur=COLORS[int(etat)]
    cnv.create_rectangle(A,B,fill=couleur,outline='')

def remplir(F):
    n=len(F)
    for x in range(n):
        for y in range(n):
            cellule(F,x,y)
def propagation():
    updateforet(F)
    cnv.delete('all')
    remplir(F)
    cnv.after(50,propagation)

p=0.7
n=100
```

1

```
48| unit=5
49| # Fenêtre et canevas
50| root = Tk()
51| cnv = Canvas(root, width=unit*n, height=unit*n,
background="ivory")
52| cnv.pack()
53| # Forêt aléatoire
54| F=Graph(n,p)
55| i=n//2
56| j=n//2
57| F[i][j]=2
58| # Plateau dessiné
59| remplir(F)
60| propagation()
61| root.mainloop()
```

2