

Biais dans les algorithmes d'apprentissage

Police prédictive

MARZOUKI Mohamed Maamoun

Session 2026

- 1 Introduction
- 2 Modélisation
- 3 Mise en évidence des biais
- 4 Remède
- 5 Annexe

L'IA dans la prise de décision publique

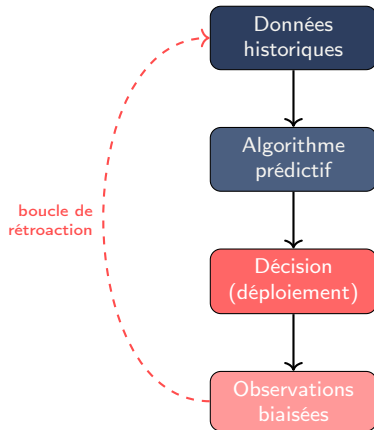
Contexte

Les algorithmes d'apprentissage automatique influencent des décisions à forts enjeux :

- Octroi de prêts, embauche
- Déploiement des forces de l'ordre

Questions éthiques

- Un algorithme *neutre* peut-il reproduire — voire amplifier — des inégalités existantes ?
- Qui est responsable quand l'algorithme discrimine ?
- Les données historiques reflètent-elles la réalité ou les biais du passé ?



Définition

La **police prédictive** consiste à utiliser les données de criminalité passées pour prévoir les zones où des crimes risquent de se produire et y orienter les forces de police.

Problème

De tels systèmes ont montré qu'ils peuvent générer des boucles de rétroaction divergentes [1], où la police est envoyée de façon répétée dans les mêmes quartiers, indépendamment du taux réel de criminalité.

Comment les données produites par un algorithme biaisent-elles son propre apprentissage ?

Sommaire

1 Introduction

2 Modélisation

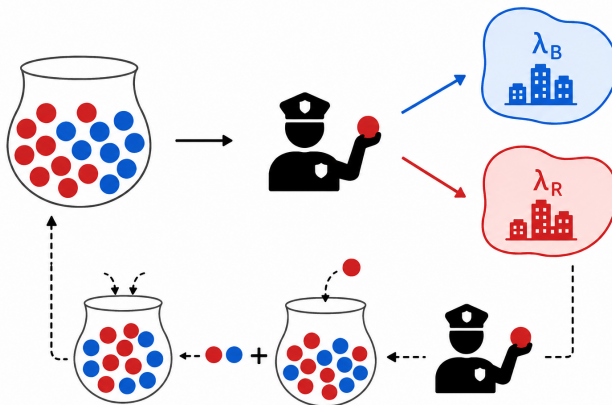
3 Mise en évidence des biais

4 Remède

5 Annexe

Modélisation

- deux quartiers R (rouge) et B (bleu)
- λ_R , λ_B : taux de criminalité sous-jacents
- urne contenant $n_R^{(t)}$ et $n_B^{(t)}$ = statistiques de criminalité $0 \rightarrow t$
- proportion de boules $i \in \{R, B\}$ à t = croyance sur taux de criminalité



Urnes de Pólya–Eggenberger [2]

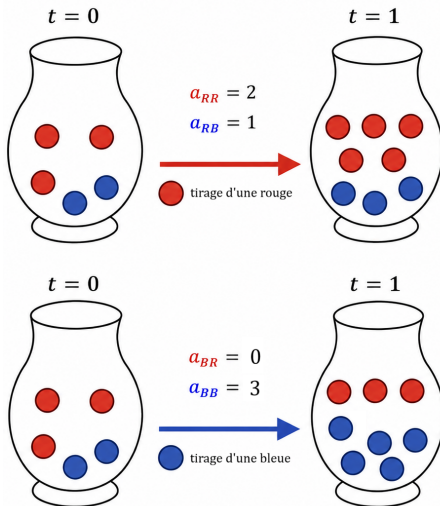


Figure: Principe d'une urne de Pólya [2]

Matrice de Remplacement

Matrice de remplacement

$$A = \begin{pmatrix} a_{RR} & a_{RB} \\ a_{BR} & a_{BB} \end{pmatrix}$$

- Si une boule rouge est tirée :
 - on ajoute a_{RR} boules rouges.
 - on ajoute a_{RB} boules bleues.
- Si une boule bleue est tirée :
 - on ajoute a_{BR} boules rouges.
 - on ajoute a_{BB} boules bleues.
- Toute boule tirée est remise.

- 1 policier choisit R ou B selon urne
- $i \in \{R, B\}$ choisi, incident/crime détecté avec d_i , +1 boule i
- incident signalé avec s_i , $\perp$ visité ou non
- Poids: π_s et π_d avec $\pi_s + \pi_d = 1$

Hypothèse (H)

$s_i = d_i = \lambda_i$ Permet au système de police prédictive d'opérer dans un cadre favorable. On verra que, même dans ce cadre optimiste, des problèmes apparaissent.

Sommaire

- 1 Introduction
- 2 Modélisation
- 3 Mise en évidence des biais**
- 4 Remède
- 5 Annexe

Incidents découverts uniquement

Cas particulier (taux de criminalité homogène)

- $\lambda_R = \lambda_B = \lambda$
- matrice de remplacement $\equiv \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ "ralentie" d'un facteur λ

Théorème (David A. Freedman [4])

$$\begin{pmatrix} \alpha & 0 \\ 0 & \alpha \end{pmatrix}, \quad \alpha > 0.$$

$$X_n = \frac{R_n}{R_n + B_n}$$

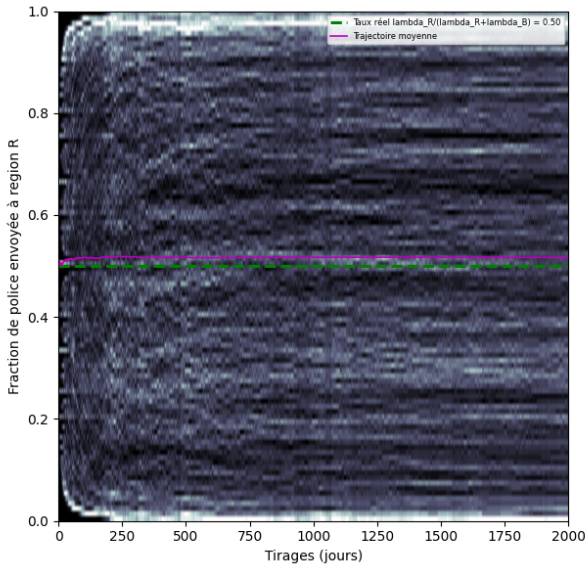
converge p.s vers une variable aléatoire X de loi bêta :

$$X \sim \text{Beta}\left(\frac{R_0}{\alpha}, \frac{B_0}{\alpha}\right).$$

Taux de criminalité homogène

- La probabilité à long terme d'obtenir une boule rouge correspond à l'estimation du taux de criminalité dans la zone R donnée par le modèle.
- Le résultat précédent montre que cette probabilité est aléatoire et dépend seulement des paramètres R_0 et B_0 qui représentent la croyance initiale du système.
- Autrement dit, cette croyance de départ, combinée au manque d'informations sur la zone non observée, empêche le système de comprendre que les deux zones ont en réalité le même taux de criminalité.

Taux de criminalité homogène



Mise en évidence des biais

Cas général

- Les mises à jour de l'urne sont désormais des variables aléatoires plutôt que des ajouts déterministes.

-

$$\begin{pmatrix} X_R & 0 \\ 0 & X_B \end{pmatrix}$$

$$X_R \sim \text{Bernoulli}(\lambda_R)$$

$$X_B \sim \text{Bernoulli}(\lambda_B)$$

Mise en évidence des biais

- L'urne ne vérifie pas la condition dite d'**équilibre**, les résultats classiques permettant de déterminer le comportement limite, ne sont pas applicables!
- Cependant, il est possible de ramener ce modèle à mises à jour aléatoires à un modèle à mises à jour déterministes en exploitant la structure de Bernoulli des ajouts.

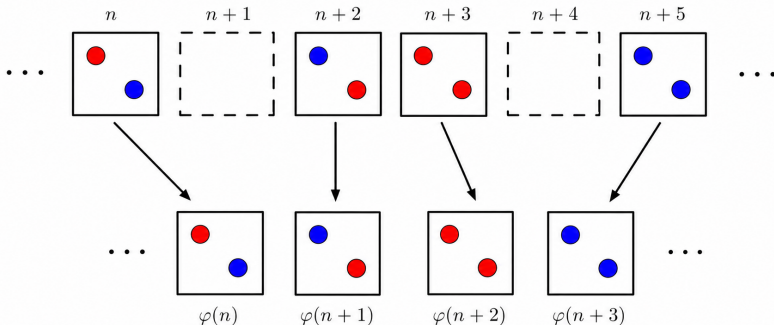


Figure: Extraction d'une sous-suite active du processus d'urne

Mise en évidence des biais

Réduction à une urne déterministe

- À chaque étape, une boule est ajoutée avec probabilité :

$$P(\text{ajout}) = \frac{n_R^{(t)} \lambda_R + n_B^{(t)} \lambda_B}{n_R^{(t)} + n_B^{(t)}}$$

- **Conditionnellement** à un ajout, la boule est rouge avec probabilité :

$$P(\text{rouge} \mid \text{ajout}) = \frac{n_R^{(t)} \lambda_R}{n_R^{(t)} \lambda_R + n_B^{(t)} \lambda_B}$$

- Dans une urne déterministe de matrice $\begin{pmatrix} \lambda_R & 0 \\ 0 & \lambda_B \end{pmatrix}$, la probabilité de tirer rouge et d'ajouter une boule rouge vaut :

$$\frac{n_R^{(t)}}{n_R^{(t)} + n_B^{(t)}} \cdot \lambda_R \Rightarrow P(\text{rouge} \mid \text{ajout}) = \frac{n_R^{(t)} \lambda_R}{n_R^{(t)} \lambda_R + n_B^{(t)} \lambda_B}$$

- Expressions **identiques** sur la sous-suite active, l'urne aléatoire et l'urne déterministe ont le même comportement limite.

Mise en évidence des biais

Théorème (Henrik Renlund [5])

Considérons une urne de Pólya de matrice de remplacement

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

$a, b, c, d \geq 0$ non tous nuls.

Alors la limite de la proportion de boules de chaque type existe presque sûrement. La proportion p de boules R peut être caractérisée comme suit :

- $a = d$ et $b = c = 0 \implies p$ cvg loi bêta. (Freedman)
- Sinon p converge vers x^* , où $x^* \in [0, 1]$ est racine de

$$P = (c + d - a - b)X^2 + (a - 2c - d)X + c$$

$$P'(x^*) < 0.$$

Mise en évidence des biais

Lemme

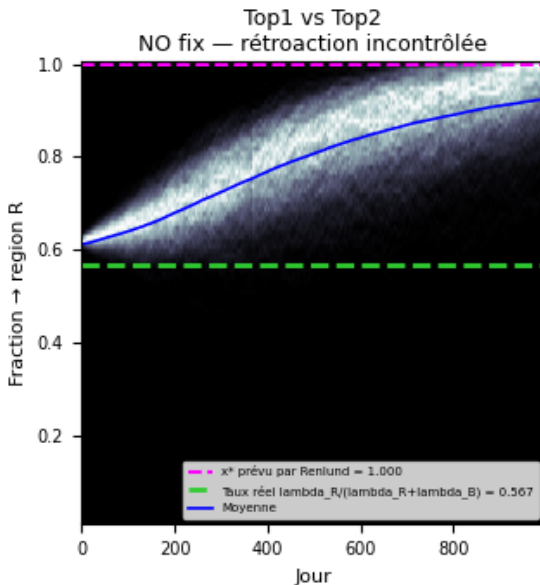
Supposons que seuls les incidents découverts sont pris en compte ($\pi_d = 1$, $\pi_s = 0$) et que $\lambda_R \neq \lambda_B$. Alors la proportion de boules rouges X_n converge presque sûrement vers 0 ou 1 :

$$X_n \xrightarrow{\text{p.s.}} \begin{cases} 1 & \text{si } \lambda_R > \lambda_B \\ 0 & \text{si } \lambda_R < \lambda_B \end{cases}$$

Signification

Dans ce scénario, le processus de mise à jour considérera qu'une région présente beaucoup plus de criminalité que l'autre, même si les taux de criminalité sont similaires.

Mise en évidence des biais



Prise en compte des incidents signalés

Matrice de remplacement — cas général

$$A = \begin{pmatrix} \pi_d \cdot d_R + \pi_s \cdot s_R & \pi_s \cdot s_B \\ \pi_s \cdot s_R & \pi_d \cdot d_B + \pi_s \cdot s_B \end{pmatrix}$$

- Diagonale : incidents **découverts** ($\pi_d \cdot d_i$) + incidents **signalés** du même quartier ($\pi_s \cdot s_i$)
- Hors-diagonale : incidents **signalés** de l'autre quartier ($\pi_s \cdot s_j$), indépendants de la patrouille

Remarque: cas particulier déjà étudié

Sous l'hypothèse (III) et $\pi_d = 1$, $\pi_s = 0$:

$$A = \begin{pmatrix} \lambda_R & 0 \\ 0 & \lambda_B \end{pmatrix}$$

Modèle général

Matrice de remplacement

Sous hypothèse ($\mathbb{H}$) ($d_i = s_i = \lambda_i$) et $\pi_d + \pi_s = 1$:

$$\begin{pmatrix} \pi_d \lambda_R + \pi_s \lambda_R & \pi_s \lambda_B \\ \pi_s \lambda_R & \pi_d \lambda_B + \pi_s \lambda_B \end{pmatrix} = \begin{pmatrix} \lambda_R & \pi_s \lambda_B \\ \pi_s \lambda_R & \lambda_B \end{pmatrix}$$

Polynôme associé (Renlund)

Avec $a = \lambda_R$, $b = \pi_s \lambda_B$, $c = \pi_s \lambda_R$, $d = \lambda_B$:

$$P(X) = \delta_d X^2 - (\delta_d + S) X + \pi_s \lambda_R$$

où $\delta_d = \pi_d(\lambda_B - \lambda_R)$ et $S = \pi_s(\lambda_R + \lambda_B)$

Racine stable

Résolution de $P(x) = 0$

En posant $\mu = \frac{1}{2} + \frac{S}{2\delta_d}$:

$$x = \mu \pm \sqrt{\mu^2 - \frac{\pi_s \lambda_R}{\delta_d}}$$

Puisque $P'(x) = 2\delta_d x - (\delta_d + S) < 0$ ssi $x < \mu$, la racine stable est :

$$x^* = \mu - \sqrt{\mu^2 - \frac{\pi_s \lambda_R}{\delta_d}}$$

Remarque: $\pi_s = 0$ (découverts uniquement)

$S = 0$, $x^* = 0$ ou $1 \Rightarrow$ on retrouve le lemme. ✓

Cas sans feedback : $\pi_d = 0$

Si $\pi_d = 0$, la matrice de remplacement est fixe : quel que soit le tirage, on ajoute toujours s_R boules R et s_B boules B. La limite est alors :

$$\tilde{x} = \frac{s_R}{s_R + s_B}$$

C'est la réponse attendue compte tenu des taux de signalement — le **taux cible** que le système devrait atteindre.

Changement de variable : $\phi = S/\delta_d$

On pose $\phi = \frac{S}{\delta_d} = \frac{\pi_s(\lambda_R + \lambda_B)}{\pi_d(\lambda_B - \lambda_R)}$, ce qui permet de réécrire μ :

$$\mu = \frac{1}{2} + \frac{S}{2\delta_d} = \frac{1 + \phi}{2}$$

Et

$$\frac{\pi_s \lambda_R}{\delta_d} = \tilde{x} \cdot \phi$$

x^* en fonction de $\tilde{x}$

Expression de x^*

L'équation se réécrit :

$$x^* = \frac{1 + \phi}{2} - \sqrt{\left(\frac{1 + \phi}{2}\right)^2 - \tilde{x}\phi}$$

- Si $\tilde{x} \rightarrow 0$: $x^* \rightarrow 0$ ✓
- Si $\tilde{x} \rightarrow 1$: $x^* \rightarrow 1$ ✓

Taux sont fortement asymétriques $\Rightarrow$ urne cvg vers bonne approx du vrai taux.

Développement limité (ϕ grand)

Approximation quand ϕ grand

Pour les valeurs intermédiaires de $\tilde{x}$, on factorise :

$$x^* = \frac{1 + \phi}{2} \left(1 - \sqrt{1 - \frac{\tilde{x}\phi}{\left(\frac{1+\phi}{2}\right)^2}} \right)$$

Pour ϕ grand, par approximation $\sqrt{1 - u} \approx 1 - u/2$:

$$x^* \approx \frac{1 + \phi}{2} \cdot \frac{\tilde{x}\phi}{2\left(\frac{1+\phi}{2}\right)^2} = \tilde{x} \cdot \frac{\phi}{\phi + 1}$$

$$x^* \approx \tilde{x} \cdot \frac{S}{S + \delta_d}$$

Signification de l'approximation

Condition pour $x^* \approx \tilde{x}$

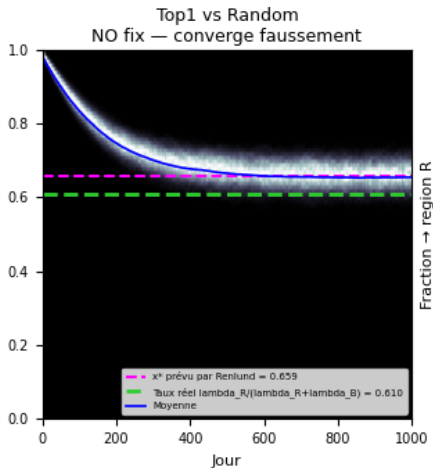
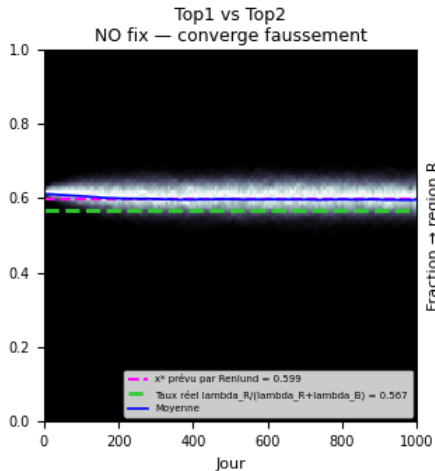
Il faut que $\frac{S}{S + \delta_d} \approx 1$, i.e. $\delta_d \ll S$ ($1 \ll \phi$). Or $\delta_d = \pi_d(\lambda_B - \lambda_R)$. Deux cas seulement :

- $\pi_d \approx 0$: on ignore les incidents découverts — le feedback disparaît mais on perd de l'information
- $\lambda_B \approx \lambda_R$: les taux sont similaires — mais c'est précisément quand ils **diffèrent** que la police prédictive est utile

Conclusion

Les seuls scénarios où le feedback ne biaise pas le résultat sont ceux où la police prédictive est soit **inutile** (taux similaires), soit **aveugle** (on ignore les découvertes). Une intervention externe reste indispensable.

Mise en évidence des biais



Sommaire

- 1 Introduction
- 2 Modélisation
- 3 Mise en évidence des biais
- 4 Remède**
- 5 Annexe

Remède — Horvitz-Thompson [3]

Étape 1 — Un 2^e tirage

Après avoir découvert un crime en R, on tire **une 2^e boule** dans l'urne.

- Si la 2^e boule est **bleue** → on ajoute la boule rouge ✓
- Si la 2^e boule est **rouge** → on **rejette** l'incident

Étape 2 — La nouvelle probabilité

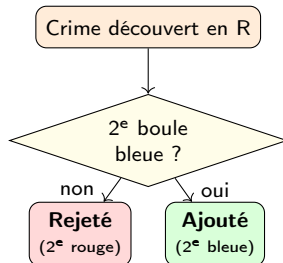
$$\begin{aligned} P(\text{aj rouge}) &= \underbrace{\frac{n_R}{n_R + n_B}}_{\text{aller en R}} \times \lambda_R \times \underbrace{\frac{n_B}{n_R + n_B}}_{\text{2^e boule bleue}} \\ &= \frac{n_R \cdot n_B}{(n_R + n_B)^2} \lambda_R \end{aligned}$$

Étape 3 — Pourquoi ça marche

Le **ratio** des deux probabilités vaut :

$$\frac{P(\text{ajout rouge})}{P(\text{ajout bleu})} = \frac{\lambda_R}{\lambda_B}$$

Seuls les vrais taux de criminalité déterminent la croissance de l'urne. La boule est **brisée**. □



Paramètres de simulation

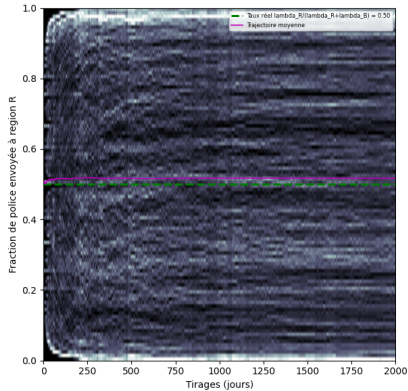
Deux raffinements clés

- **Loi de Poisson** : nombre de crimes journaliers aléatoire, modélise la variabilité réelle. La boucle de rétroaction persiste indépendamment du bruit.
- **Décroissance exponentielle** : chaque boule disparaît avec probabilité p_d à chaque étape — inspiré du *pheromone decay* en optimisation par colonies de fourmis (ACO). Les données anciennes perdent progressivement leur influence.

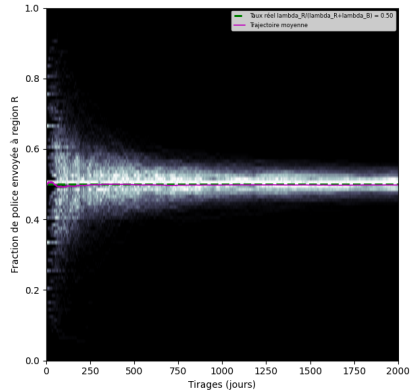
Valeurs utilisées

$p_d = 0.01$, mises à jour $\sim \text{Poisson}(\lambda_i)$, 1000 runs, 1000 jours chacun.

Simulation — Taux homogène



Sans correction



Avec correction

Simulation — Découverts Uniquement

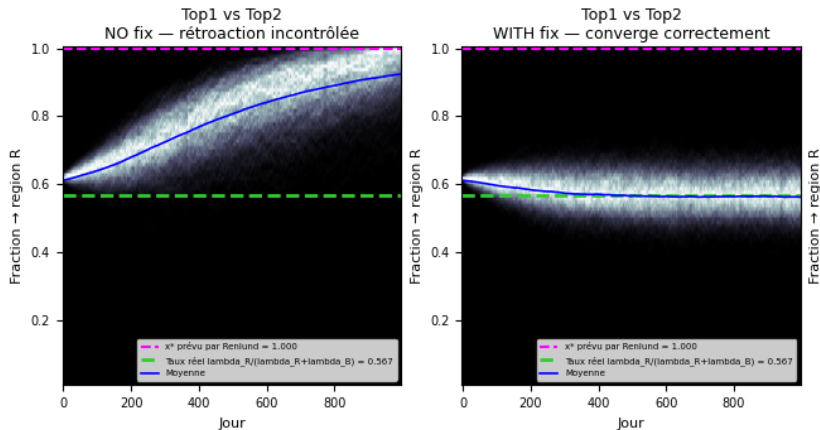


Figure: DEC (1v2)

Simulation — Découverts Uniquement

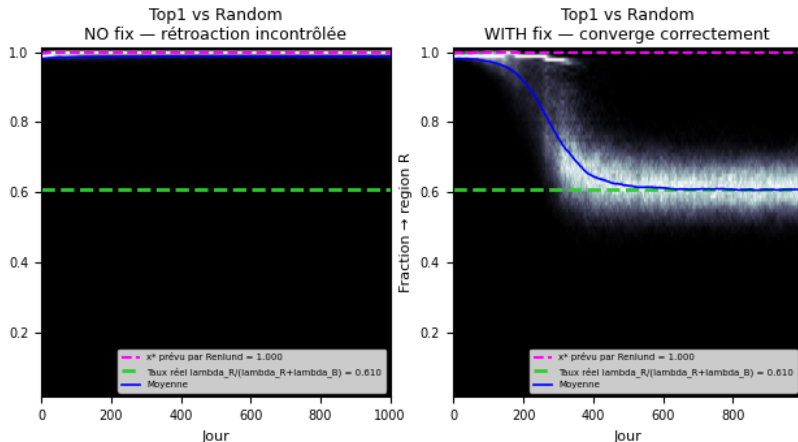


Figure: DEC (1vR)

Simulation — Découverts et Signalés

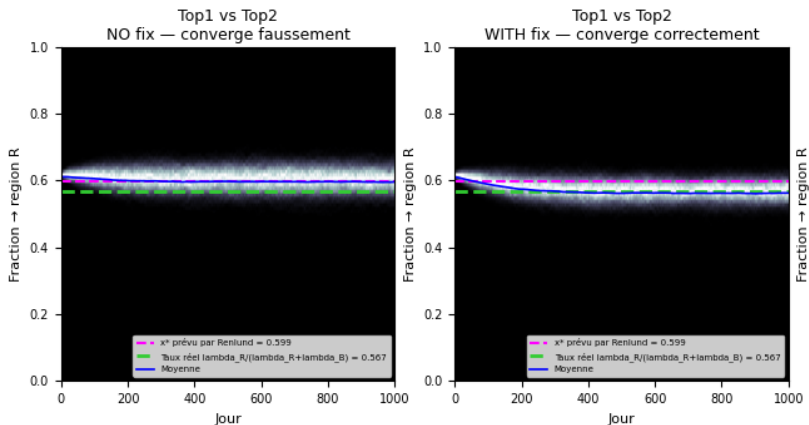


Figure: ALL(1v2)

Simulation — Découverts et Signalés

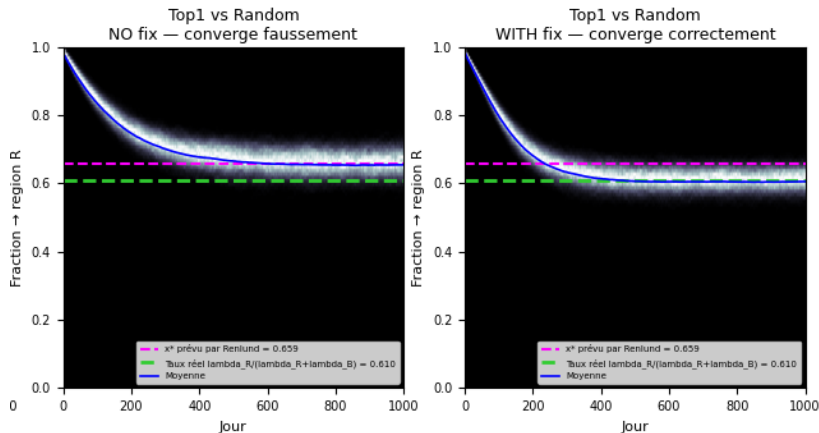


Figure: ALL(1vR)

Merci pour votre attention!

Sommaire

- 1 Introduction
- 2 Modélisation
- 3 Mise en évidence des biais
- 4 Remède
- 5 Annexe**

$$Z \sim \text{Beta}(\alpha, \beta)$$

$$P(Z = x) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1} (1-x)^{\beta-1} 1_{[0,1]}(x)$$

$$B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha + \beta)}$$

Preuve (Lemme)

- Matrice de remplacement diagonale : $\begin{pmatrix} \lambda_R & 0 \\ 0 & \lambda_B \end{pmatrix}$, avec $\lambda_R \neq \lambda_B$
- Par réduction à la sous-suite active, Renlund est applicable avec $a = \lambda_R$, $b = 0$, $c = 0$, $d = \lambda_B$
- Le polynôme associé devient :

$$P(X) = (\lambda_B - \lambda_R)X^2 + (\lambda_R - \lambda_B)X = (\lambda_B - \lambda_R)X(X - 1)$$

- Les racines sont $x^* = 0$ et $x^* = 1$. La condition $P'(x^*) < 0$ sélectionne :
 - $x^* = 1$ si $\lambda_R > \lambda_B$ (car $P'(1) = \lambda_B - \lambda_R < 0$... seule racine stable)
 - $x^* = 0$ si $\lambda_R < \lambda_B$
- $\Rightarrow$ la boucle de rétroaction envoie **toute** la police dans le quartier au taux dominant $\square$

Décroissance & Analogie ACO

Décroissance dans l'urne

À chaque étape, chaque boule disparaît indépendamment avec probabilité p_d :
Le poids d'un événement à $t = 0$ décroît comme $(1 - p_d)^t$.

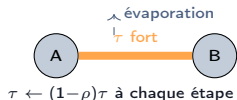
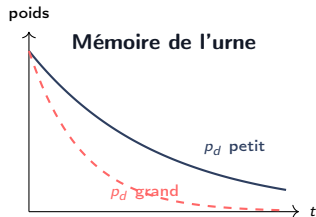
Pheromone decay (ACO)

En **Ant Colony Optimization**, les fourmis déposent des phéromones sur les bons chemins ; celles-ci s'évaporent :

$$\tau_{ij}^{(t+1)} = (1 - \rho) \tau_{ij}^{(t)} + \Delta \tau_{ij}$$

$\rho \in [0, 1]$: taux d'évaporation $\leftrightarrow p_d$
dans notre modèle.

Effet commun : empêcher la convergence prématurée vers un chemin (ou quartier) sous-optimal.



```

1 import random
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import matplotlib.colors as mcolors
5 norm = mcolors.PowerNorm(gamma=1)
6
7 LAMBDA_R = 0.5
8 LAMBDA_B = 0.5
9 NR = 1
10 NB = 1
11 NDRAWS = 2000
12 NRUNS = 500
13 a_PARAM = LAMBDA_R
14 b_PARAM = 0.0
15 c_PARAM = 0.0
16 d_PARAM = LAMBDA_B
17
18 def draw_step_no_fix(state):
19     red, blue = state
20     total = red + blue
21     v = random.random() * total
22     visited_R = (v < red)
23     if visited_R:
24         new_red = np.random.poisson(a_PARAM)
25         new_blue = 0
26     else:
27         new_red = 0
28         new_blue = np.random.poisson(d_PARAM)
29     return (red + new_red, blue + new_blue)
30
31 def draw_step_with_fix(state):
32     red, blue = state
33     total = red + blue
34     v = random.random() * total
35     visited_R = (v < red)
36     surprise_R = blue / total
37     surprise_B = red / total
38     max_surprise = max(surprise_R, surprise_B) #Normalise
39     surprise_R /= max_surprise #minority region surprise < 1 (never rejected)
40     surprise_B /= max_surprise #majority region surprise < 1 (sometimes rejected)
41     r = random.random()
42     if visited_R:
43         if r >= surprise_R:
44             return state # crime record rejected
45         new_red = np.random.poisson(a_PARAM)
46         new_blue = 0
47     else:
48         if r >= surprise_B:
49             return state # crime record rejected
50         new_red = 0
51         new_blue = np.random.poisson(d_PARAM)
52     return (red + new_red, blue + new_blue)

```

```

1 def run_simulation(draw_step_fn):
2     histo_resolution = 100 # 101 cases de 0% à 100%
3     counts = np.zeros((histo_resolution + 1, NDRAWS)) #counts[x,y]=N' de runs passant de
4     (x,y)
5     for run in range(NRUNS):
6         state = (NR, NB)
7         for day in range(NDRAWS):
8             state = draw_step_fn(state)
9             red, blue = state
10            frac = red / (red + blue)
11            bin_idx = 100 - int(frac * histo_resolution) # (100 - ...) pour avoir y=1.0 en
12            haut
13            counts[bin_idx, day] += 1
14            bin_values = np.linspace(1.0, 0.0, histo_resolution + 1)
15            v = (counts * bin_values[:, np.newaxis]).sum(axis=0) / counts.sum(axis=0)
16            counts_norm = counts / counts.max(axis=0) #Normaliser (gradient de couleur)
17            return counts_norm, v
18
19 def plot_result(ax, counts_norm, v, title=""):
20     ndraws = counts_norm.shape[1]
21     ax.imshow(counts_norm, extent=[0, ndraws, 0, 1.0],
22              aspect=ndraws, cmap="bone",
23              origin="upper", norm=norm)
24
25     true_frac = LAMBDA_R / (LAMBDA_R + LAMBDA_B)
26     ax.plot([0, ndraws], [true_frac, true_frac], "g--", linewidth=2.2,
27            label=f"Taux réel lambda_R/(lambda_R+lambda_B) = {true_frac:.2f}")
28
29     ax.plot(np.arange(ndraws), v, "m-", lw=1.2, label="Trajectoire moyenne")
30     ax.set_xlim([0, ndraws])
31     ax.set_ylim([0, 1])
32     ax.set_xlabel("Tirages (jours)")
33     ax.set_ylabel("Fraction de police envoyée à région R")
34     ax.legend(fontsize=6, loc="upper right")
35
36 counts_no_fix, v_no_fix = run_simulation(draw_step_no_fix)
37 counts_fix, v_fix = run_simulation(draw_step_with_fix)
38
39 fig, (ax_left, ax_right) = plt.subplots(1, 2, figsize=(16, 6))
40 plot_result(ax_left, counts_no_fix, v_no_fix)
41 plot_result(ax_right, counts_fix, v_fix)
42 plt.tight_layout() #ajuste "spacing" entre les subplots (no overlap)
43 plt.show()

```

```

1 import random
2 import numpy as np
3 import matplotlib.pyplot as plt
4 LAMBDA_TOP1 = 3.685
5 LAMBDA_TOP2 = 2.816
6 LAMBDA_RANDOM = 2.368
7 INIT_TOP1 = 609
8 INIT_TOP2 = 379
9 INIT_RANDOM = 7
10 NDRAWS = 1000
11 NRUNS = 1000
12 DECAY = 0.01
13 PD, PS = 0.5, 0.5
14 def make_matrix_discovered_only(lambda_R, lambda_B):
15     return [[lambda_R, 0.0],
16             [0.0, lambda_B]]
17 def make_matrix_all_incidents(lambda_R, lambda_B, pd=PD, ps=PS):
18     return [[lambda_R, ps + lambda_B],
19             [ps + lambda_R, lambda_B]]
20 def poly_a_draw(initial_state, update_matrix,
21                 decay=0.0, use_surprise=False,
22                 use_partial_surprise=None, n_draws=NDRAWS):
23     binom = np.random.binomial
24     poisson = np.random.poisson
25     rr = random.random
26     ss = initial_state
27     for _ in range(n_draws):
28         n_red, n_blue = ss
29         total = n_red + n_blue
30         choice = 0 if rr() + total < n_red else 1
31         ur = [update_matrix[choice][0], update_matrix[choice][1]]
32         ur = [float(poisson(ur[0])), float(poisson(ur[1]))]
33         if use_surprise:
34             p_r, p_b = n_red / total, n_blue / total
35             v = [p_b, p_r]
36             mx = max(v)
37             v = [v[0] / mx, v[1] / mx]
38             if rr() >= v[choice]:
39                 if use_partial_surprise is not None:
40                     ur = list(use_partial_surprise)
41                 else:
42                     ur = [0.0, 0.0]
43             ss = (n_red + ur[0], n_blue + ur[1])
44         if decay > 0.0:
45             d = binom(ss, decay)
46             ss = (max(ss[0] - d[0], 1.0), max(ss[1] - d[1], 1.0))
47     yield ss

```

```

1 def penlund_root(a, b, c, d):
2     A2 = c + d - a - b # coefficient de X2
3     A1 = a - 2*c - d # coefficient de X
4     AB = c # terme constant
5     if abs(A2) < 1e-12: # 1e-12 est 10**(-12) (eviter erreurs d'arrondi flottant)
6         if abs(A1) < 1e-12: # 10**(-12) (precision de 10**(-16) (64-bit des floats python)
7             return None
8         root = -AB / A1
9         return root if 0 <= root <= 1 else None
10    discriminant = A1**2 - 4 * A2 * AB
11    if discriminant < 0:
12        return None
13    sqrt_disc = np.sqrt(discriminant)
14    x1 = (-A1 - sqrt_disc) / (2 * A2)
15    x2 = (-A1 + sqrt_disc) / (2 * A2)
16    def f_prime(x):
17        return 2 * A2 + x + A1
18    stable_root = None
19    for root in [x1, x2]:
20        if 0 <= root <= 1 and f_prime(root) < 0:
21            stable_root = root
22            break
23    if stable_root is None:
24        return None
25    return stable_root
26
27 def run_simulation(initial_state, update_matrix,
28                  decay=DECAY,
29                  use_surprise=False,
30                  use_partial_surprise=None,
31                  ndraws=NDRAWS, nruns=NRUNS,
32                  label=""):
33     histo_res = 100
34     counts = np.zeros((histo_res + 1, ndraws))
35     for run_1 in range(nruns):
36         for day, state in enumerate(
37             poly_a_draw(initial_state, update_matrix,
38                         decay=decay,
39                         use_surprise=use_surprise,
40                         use_partial_surprise=use_partial_surprise,
41                         n_draws=ndraws)):
42             frac = state[0] / (state[0] + state[1])
43             bin_idx = histo_res - int(frac * histo_res)
44             bin_idx = min(bin_idx, histo_res)
45             counts[bin_idx, day] += 1
46     bin_values = np.linspace(1.0, 0.0, histo_res + 1)
47     mean_traj = (counts + bin_values[:, np.newaxis]).sum(axis=0) / counts.sum(axis=0)
48     col_max = counts.max(axis=0)
49     col_max[col_max == 0] = 1
50     counts_norm = counts / col_max
51     ((a, b), (c, d)) = update_matrix
52     root = penlund_root(a, b, c, d)
53     return counts_norm, mean_traj, root

```

```

Simulation
1 def draw_panel(ax, counts_norm, mean_traj, root,
2               true_rate_frac, ndraws, title):
3     ax.imshow(counts_norm,
4               extent=[0, ndraws, 0, 1.01],
5               aspect=ndraws,
6               cmap="bone",
7               origin="upper",
8               vmin=0, vmax=1)
9     if np.isreal(root) and 0.0 <= root.real <= 1.0:
10         ax.axhline(root.real, color="magenta", linestyle="--", linewidth=1.5,
11                   label=f"x* prévu par RenLund = {root.real:.3f}")
12         ax.axhline(true_rate_frac, color="limegreen", linestyle="--", linewidth=2,
13                   label=f"Taux réel lambda_R/(lambda_R+lambda_B) = {true_rate_frac:.3f}")
14         ax.plot(np.arange(ndraws), mean_traj, color="blue",
15                linewidth=1.2, alpha=1, label="Moyenne")
16         ax.set_xlim([0, ndraws])
17         ax.set_ylim([0, 1])
18         ax.set_xlabel("Jour", fontsize=8)
19         ax.set_ylabel("Fraction → region R", fontsize=8)
20         ax.set_title(title, fontsize=9, pad=4)
21         ax.legend(fontsize=5, loc="lower right")
22         ax.tick_params(labelsize=7)
23
24 def run_all():
25     mat_tit2_disc = make_matrix_discovered_only(LAMBDA_TOP1, LAMBDA_TOP2)
26     mat_tirand_disc = make_matrix_discovered_only(LAMBDA_TOP1, LAMBDA_RANDOM)
27
28     mat_tit2_all = make_matrix_all_incidents(LAMBDA_TOP1, LAMBDA_TOP2)
29     mat_tirand_all = make_matrix_all_incidents(LAMBDA_TOP1, LAMBDA_RANDOM)
30
31     frac_tit2 = LAMBDA_TOP1 / (LAMBDA_TOP1 + LAMBDA_TOP2) # ≈ 0.567
32     frac_tirand = LAMBDA_TOP1 / (LAMBDA_TOP1 + LAMBDA_RANDOM) # ≈ 0.610
33
34     partial_tit2_all = (PS + LAMBDA_TOP1, PS + LAMBDA_TOP2)
35     partial_tirand_all = (PS + LAMBDA_TOP1, PS + LAMBDA_RANDOM)
36
37     state_tit2 = (float(INIT_TOP1), float(INIT_TOP2))
38     state_tirand = (float(INIT_TOP1), float(INIT_RANDOM))
39
40     r1 = run_simulation(state_tit2, mat_tit2_disc,
41                       use_surprise=False, decay=DECAY, ndraws=NDRAWS, nruns=NRUNS)
42
43     r2 = run_simulation(state_tit2, mat_tit2_disc,
44                       use_surprise=True, decay=DECAY, ndraws=NDRAWS, nruns=NRUNS)
45
46     r3 = run_simulation(state_tirand, mat_tirand_disc,
47                       use_surprise=False, decay=DECAY, ndraws=NDRAWS, nruns=NRUNS)
48
49     r4 = run_simulation(state_tirand, mat_tirand_disc,
50                       use_surprise=True, decay=DECAY, ndraws=NDRAWS, nruns=NRUNS)

```

```

Simulation
1     r5 = run_simulation(state_tit2, mat_tit2_all,
2                       use_surprise=False, decay=DECAY, ndraws=NDRAWS, nruns=NRUNS)
3
4     r6 = run_simulation(state_tit2, mat_tit2_all,
5                       use_surprise=True,
6                       use_partial_surprise=partial_tit2_all,
7                       decay=DECAY, ndraws=NDRAWS, nruns=NRUNS)
8
9     r7 = run_simulation(state_tirand, mat_tirand_all,
10                      use_surprise=False, decay=DECAY, ndraws=NDRAWS, nruns=NRUNS)
11
12     r8 = run_simulation(state_tirand, mat_tirand_all,
13                      use_surprise=True,
14                      use_partial_surprise=partial_tirand_all,
15                      decay=DECAY, ndraws=NDRAWS, nruns=NRUNS)
16
17     fig1, axes1 = plt.subplots(1, 4, figsize=(18, 5))
18
19     draw_panel(axes1[0], r1, frac_tit2, NDRAWS,
20               "Top1 vs Top2|nNO fix - rétroaction incontrôlée")
21
22     draw_panel(axes1[1], r2, frac_tit2, NDRAWS,
23               "Top1 vs Top2|nMITH fix - converge correctement")
24
25     draw_panel(axes1[2], r3, frac_tirand, NDRAWS,
26               "Top1 vs Random|nNO fix - rétroaction incontrôlée")
27
28     draw_panel(axes1[3], r4, frac_tirand, NDRAWS,
29               "Top1 vs Random|nMITH fix - converge correctement")
30
31     plt.tight_layout()
32     fig2, axes2 = plt.subplots(1, 4, figsize=(18, 5))
33
34     draw_panel(axes2[0], r5, frac_tit2, NDRAWS,
35               "Top1 vs Top2|nNO fix - converge faussement")
36
37     draw_panel(axes2[1], r6, frac_tit2, NDRAWS,
38               "Top1 vs Top2|nMITH fix - converge correctement")
39
40     draw_panel(axes2[2], r7, frac_tirand, NDRAWS,
41               "Top1 vs Random|nNO fix - converge faussement")
42
43     draw_panel(axes2[3], r8, frac_tirand, NDRAWS,
44               "Top1 vs Random|nMITH fix - converge correctement")
45
46     plt.tight_layout()
47     plt.show()
48
49 run_all()

```

Références bibliographiques

-  KRISTIAN LUM AND WILLIAM ISAAC : To predict and serve?
-  F. EGGENBERGER AND G. PÓLYA : Über die Statistik verketteter Vorgänge.
-  DANIEL G. HORVITZ AND DONOVAN J. THOMPSON : A generalization of sampling without replacement from a finite universe.
-  DAVID A. FREEDMAN : Bernard Friedman's Urn.
-  HENRIK RENLUND : Generalized Pólya urns via stochastic approximation.