

TIPE: Étude mathématique du mélange de cartes

Damien JUNGER

Numéro 16816
~

Épreuve de TIPE

Session 2024
MP - Option Informatique

Plan de l'exposé

- 1 Angles d'approche
- 2 Autour des mélanges probabilistes
- 3 Sur le phénomène de *cut-off*
- 4 Conclusion
- 5 Annexe

Angles d'approche

Quelles mathématiques employées ?

- Mélange parfait et déterministe : une approche algébriste (le *faro shuffle*)



- Mélange laissant place à l'aléatoire : une approche probabiliste (le *riffle shuffle*)

Angles d'approche

Quelles mathématiques employées ?

- Mélange parfait et déterministe : une approche algébrique (le *faro shuffle*)



- Mélange laissant place à l'aléatoire : une approche probabiliste (le *riffle shuffle*)



Angles d'approche

Sur le *faro shuffle*

- Seulement deux mélanges : le *in-shuffle* (noté I) et le *out-shuffle* (noté O).
- L'étude se porte alors sur le groupe engendré $\langle I, O \rangle$.
- Il existe alors des résultats (d'isomorphisme notamment) sur ce groupe. Par exemple, pour $2n = 52$ cartes, on a :

$$|\langle I, O \rangle| = n!2^n \approx 3 \times 10^{34}.$$

Angles d'approche

Sur le *faro shuffle*

- Seulement deux mélanges : le *in-shuffle* (noté I) et le *out-shuffle* (noté O).
- L'étude se porte alors sur le groupe engendré $\langle I, O \rangle$.
- Il existe alors des résultats (d'isomorphisme notamment) sur ce groupe. Par exemple, pour $2n = 52$ cartes, on a :

$$|\langle I, O \rangle| = n!2^n \approx 3 \times 10^{34}.$$

Angles d'approche

Sur le *faro shuffle*

- Seulement deux mélanges : le *in-shuffle* (noté I) et le *out-shuffle* (noté O).
- L'étude se porte alors sur le groupe engendré $\langle I, O \rangle$.
- Il existe alors des résultats (d'isomorphisme notamment) sur ce groupe. Par exemple, pour $2n = 52$ cartes, on a :

$$|\langle I, O \rangle| = n!2^n \approx 3 \times 10^{34}.$$

Angles d'approche

Sur le *faro shuffle*

- Seulement deux mélanges : le *in-shuffle* (noté I) et le *out-shuffle* (noté O).
- L'étude se porte alors sur le groupe engendré $\langle I, O \rangle$.
- Il existe alors des résultats (d'isomorphisme notamment) sur ce groupe. Par exemple, pour $2n = 52$ cartes, on a :

$$|\langle I, O \rangle| = n!2^n \approx 3 \times 10^{34}.$$

Angles d'approche

Sur le *faro shuffle*

- Seulement deux mélanges : le *in-shuffle* (noté I) et le *out-shuffle* (noté O).
- L'étude se porte alors sur le groupe engendré $\langle I, O \rangle$.
- Il existe alors des résultats (d'isomorphisme notamment) sur ce groupe. Par exemple, pour $2n = 52$ cartes, on a :

$$|\langle I, O \rangle| = n!2^n \approx 3 \times 10^{34}.$$

Angles d'approche

Défauts et limites du *faro shuffle*

- Un seul degré de liberté probabiliste (choix de la première carte) .
- Facilement prédictible.

Angles d'approche

Défauts et limites du *faro shuffle*

- Un seul degré de liberté probabiliste (choix de la première carte) .
- Facilement prédictible.

Angles d'approche

Défauts et limites du *faro shuffle*

- Un seul degré de liberté probabiliste (choix de la première carte) .
- Facilement prédictible.

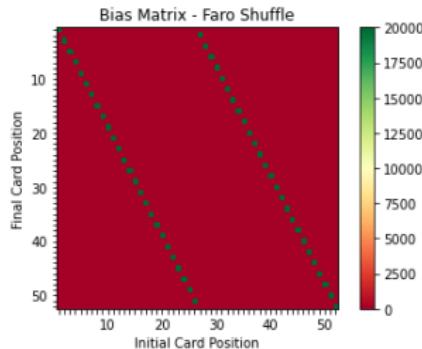


Figure – Matrice de biais du *faro shuffle*

Angles d'approche

Défaut et limite du *faro shuffle*

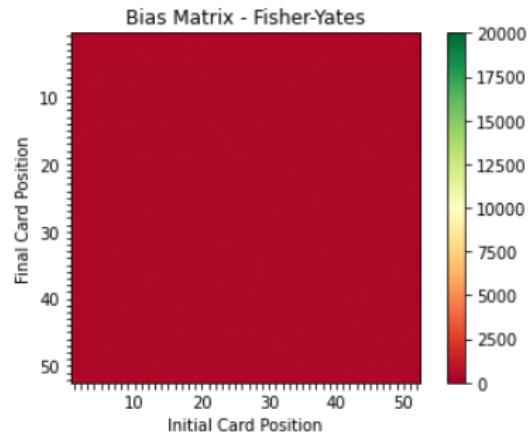
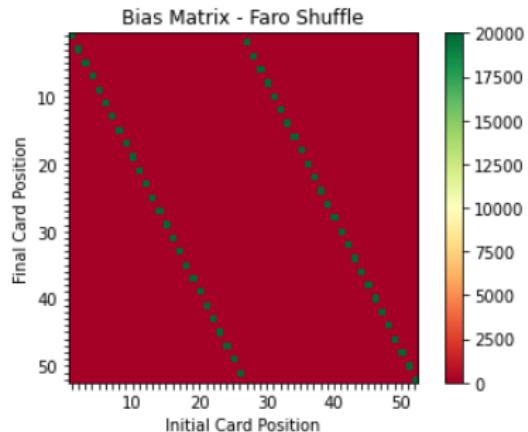


Figure – Matrice de biais du *faro shuffle*

Figure – Matrice de biais du mélange *Fisher Yates*

Présentation du modèle adopté

Hypothèse

- Objectif : rendre le *faro shuffle* moins prédictible en probabilisant, par exemple,
 - ➊ la taille des deux paquets initiaux,
 - ➋ le choix de la prochaine carte à tomber.
- On adopte alors le modèle *GSR* (pour *Gilbert-Shannon-Reeds*) :
 - ➌ la taille des paquets suit une loi binomiale de paramètre $(n, 1/2)$,
 - ➍ la j -ème carte tombe du paquet $i \in \{1, 2\}$ avec la probabilité

$$p_{i,j} = N_{i,j}/(N_{1,j} + N_{2,j}).$$

Présentation du modèle adopté

Hypothèse

- Objectif : rendre le *faro shuffle* moins prédictible en probabilisant, par exemple,
 - ➊ la taille des deux paquets initiaux,
 - ➋ le choix de la prochaine carte à tomber.
- On adopte alors le modèle *GSR* (pour *Gilbert-Shannon-Reeds*) :
 - ➌ la taille des paquets suit une loi binomiale de paramètre $(n, 1/2)$,
 - ➍ la j -ème carte tombe du paquet $i \in \{1, 2\}$ avec la probabilité

$$p_{i,j} = N_{i,j}/(N_{1,j} + N_{2,j}).$$

Présentation du modèle adopté

Hypothèse

- Objectif : rendre le *faro shuffle* moins prédictible en probabilisant, par exemple,
 - ❶ la taille des deux paquets initiaux,
 - ❷ le choix de la prochaine carte à tomber.
- On adopte alors le modèle *GSR* (pour *Gilbert-Shannon-Reeds*) :
 - ❶ la taille des paquets suit une loi binomiale de paramètre $(n, 1/2)$,
 - ❷ la j -ème carte tombe du paquet $i \in \{1, 2\}$ avec la probabilité

$$p_{i,j} = N_{i,j}/(N_{1,j} + N_{2,j}).$$

Présentation du modèle adopté

Hypothèse

- Objectif : rendre le *faro shuffle* moins prédictible en probabilisant, par exemple,
 - ➊ la taille des deux paquets initiaux,
 - ➋ le choix de la prochaine carte à tomber.
- On adopte alors le modèle *GSR* (pour *Gilbert-Shannon-Reeds*) :
 - ➊ la taille des paquets suit une loi binomiale de paramètre $(n, 1/2)$,
 - ➋ la j -ème carte tombe du paquet $i \in \{1, 2\}$ avec la probabilité

$$p_{i,j} = N_{i,j}/(N_{1,j} + N_{2,j}).$$

Présentation du modèle adopté

Hypothèse

- Objectif : rendre le *faro shuffle* moins prédictible en probabilisant, par exemple,
 - ① la taille des deux paquets initiaux,
 - ② le choix de la prochaine carte à tomber.
- On adopte alors le modèle *GSR* (pour *Gilbert-Shannon-Reeds*) :
 - ① la taille des paquets suit une loi binomiale de paramètre $(n, 1/2)$,
 - ② la j -ème carte tombe du paquet $i \in \{1, 2\}$ avec la probabilité

$$p_{i,j} = N_{i,j}/(N_{1,j} + N_{2,j}).$$

Présentation du modèle adopté

Hypothèse

- Objectif : rendre le *faro shuffle* moins prédictible en probabilisant, par exemple,
 - ① la taille des deux paquets initiaux,
 - ② le choix de la prochaine carte à tomber.
- On adopte alors le modèle *GSR* (pour *Gilbert-Shannon-Reeds*) :
 - ① la taille des paquets suit une loi binomiale de paramètre $(n, 1/2)$,
 - ② la j -ème carte tombe du paquet $i \in \{1, 2\}$ avec la probabilité

$$p_{i,j} = N_{i,j}/(N_{1,j} + N_{2,j}).$$

Le modèle *GSR*

Comparaison avec le *riffle shuffle*

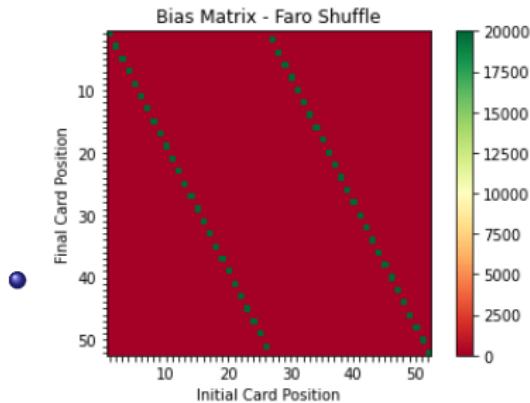


Figure – Matrice de biais du *faro shuffle*

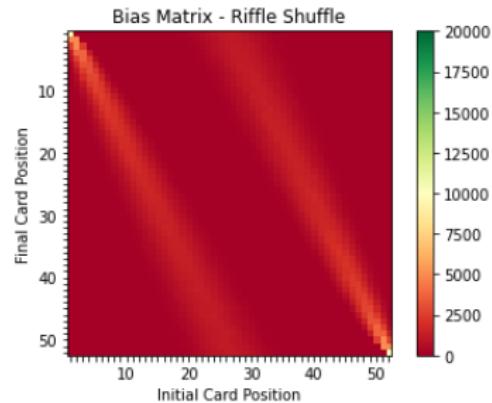


Figure – Matrice de biais du *riffle shuffle*

- Le mélange est donc plus efficace.

Le modèle *GSR*

Comparaison avec le *riffle shuffle*

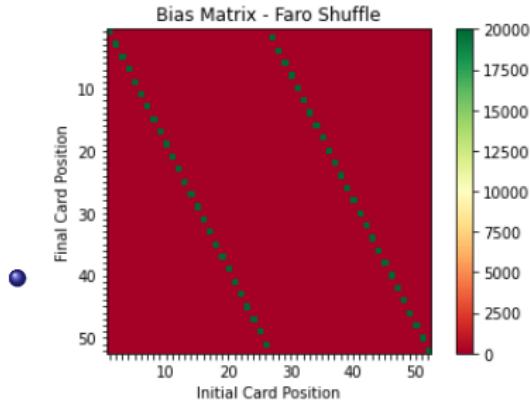


Figure – Matrice de biais du *faro shuffle*

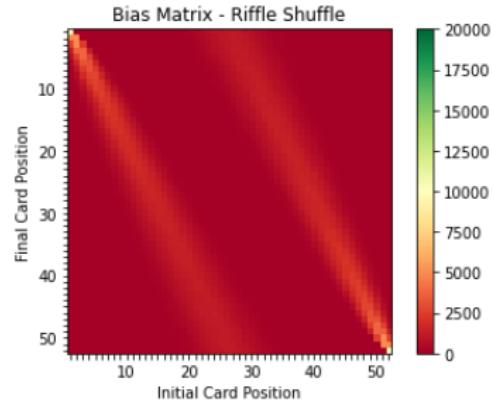


Figure – Matrice de biais du *riffle shuffle*

- Le mélange est donc plus efficace.

Le modèle *GSR*

La mesure du désordre, ou comment quantifier un *bon mélange*

L'espace probabilisé est $(\mathfrak{S}_n, \mathcal{P}(\mathfrak{S}_n), \mathbb{P})$; S_k est la variable aléatoire donnant l'état du paquet après k mélanges, de matrice de transition P ; S^∞ associée à $P^\infty := \lim_{k \rightarrow +\infty} P^k$. On définit

- L'**entropie de Shannon** H par

$$H = - \sum_{\sigma \in \mathfrak{S}_n} \mathbb{P}(S_k = \sigma) \log_2(\mathbb{P}(S_k = \sigma))$$

- La **distance de variation totale** $d_{tv}(S_k, S^\infty)$ par

$$d_{tv}(S_k, S^\infty) = \frac{1}{2} \max_{i \in [1, n]} \sum_{j=1}^n |(P^k - P^\infty)_{i,j}|$$

Le modèle *GSR*

La mesure du désordre, ou comment quantifier un *bon mélange*

L'espace probabilisé est $(\mathfrak{S}_n, \mathcal{P}(\mathfrak{S}_n), \mathbb{P})$; S_k est la variable aléatoire donnant l'état du paquet après k mélanges, de matrice de transition P ; S^∞ associée à $P^\infty := \lim_{k \rightarrow +\infty} P^k$. On définit

- L'**entropie de Shannon** H par

$$H = - \sum_{\sigma \in \mathfrak{S}_n} \mathbb{P}(S_k = \sigma) \log_2(\mathbb{P}(S_k = \sigma))$$

- La **distance de variation totale** $d_{tv}(S_k, S^\infty)$ par

$$d_{tv}(S_k, S^\infty) = \frac{1}{2} \max_{i \in \llbracket 1, n \rrbracket} \sum_{j=1}^n |(P^k - P^\infty)_{i,j}|$$

Le phénomène de *cut-off*

Étude graphique de *riffle shuffle* successifs

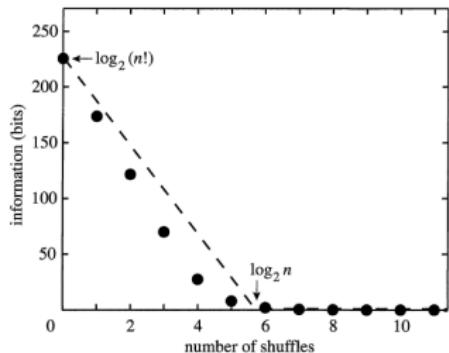


Figure – Information $I = \log_2(n!) - H$ en fonction du nombre de *riffle shuffle* successifs

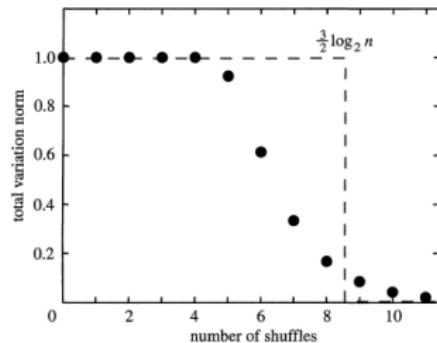


Figure – Distance de variation totale en fonction du nombre de *riffle shuffle* successifs

D'après "How many shuffles to randomize a deck of cards?", L.N.Trefethen et L.M.Trefethen

Le phénomène de *cut-off*

Apparition du *cut-off*

- Observation d'une large diminution de la variation totale de la distance au niveau de 7 mélanges pour 52 cartes : c'est le phénomène de *cut-off*.
- On peut démontrer rigoureusement ce résultat et obtenir asymptotiquement :

$$\forall \varepsilon \in \mathbb{R}^{+*}, \inf(\{k \in \mathbb{N}, d_{tv}(P^k, P^\infty) \leq \varepsilon\}) \underset{n \rightarrow +\infty}{\sim} \frac{3}{2} \log_2(n).$$

Le phénomène de *cut-off*

Apparition du *cut-off*

- Observation d'une large diminution de la variation totale de la distance au niveau de 7 mélanges pour 52 cartes : c'est le phénomène de *cut-off*.
- On peut démontrer rigoureusement ce résultat et obtenir asymptotiquement :

$$\forall \varepsilon \in \mathbb{R}^{+*}, \inf(\{k \in \mathbb{N}, d_{tv}(P^k, P^\infty) \leq \varepsilon\}) \underset{n \rightarrow +\infty}{\sim} \frac{3}{2} \log_2(n).$$

Le phénomène de *cut-off*

Questions de programmation

- Analyse fréquentielle :

- Principe : répétition d'un grand nombre d'expérience, puis calcul de la moyenne.
- Algorithme peu performant, et impossible à réaliser pour n au dessus de 10 : complexité en $\mathcal{O}(nm^2t)$.

- Analyse déterministe :

- Principe : On calcule directement les matrices itérées P^k .
- Impossible en pratique ($(52!)^2 \approx 7 \times 10^{135}$).
- Possibilité de réduction des P^k en taille n.

Le phénomène de *cut-off*

Questions de programmation

- Analyse fréquentielle :

- Principe : répétition d'un grand nombre d'expérience, puis calcul de la moyenne.
- Algorithme peu performant, et impossible à réaliser pour n au dessus de 10 : complexité en $\mathcal{O}(nm^2t)$.

- Analyse déterministe :

- Principe : On calcule directement les matrices itérées P^k .
- Impossible en pratique ($(52!)^2 \approx 7 \times 10^{135}$).
- Possibilité de réduction des P^k en taille n.

Le phénomène de *cut-off*

Questions de programmation

- Analyse fréquentielle :

- Principe : répétition d'un grand nombre d'expérience, puis calcul de la moyenne.
- Algorithme peu performant, et impossible à réaliser pour n au dessus de 10 : complexité en $\mathcal{O}(nm^2t)$.

- Analyse déterministe :

- Principe : On calcule directement les matrices itérées P^k .
- Impossible en pratique ($(52!)^2 \approx 7 \times 10^{135}$).
- Possibilité de réduction des P^k en taille n.

Le phénomène de *cut-off*

Questions de programmation

- Analyse fréquentielle :

- Principe : répétition d'un grand nombre d'expérience, puis calcul de la moyenne.
- Algorithme peu performant, et impossible à réaliser pour n au dessus de 10 : complexité en $\mathcal{O}(nm^2t)$.

- Analyse déterministe :

- Principe : On calcule directement les matrices itérées P^k .
- Impossible en pratique ($(52!)^2 \approx 7 \times 10^{135}$).
- Possibilité de réduction des P^k en taille n.

Le phénomène de *cut-off*

Questions de programmation

- Analyse fréquentielle :

- Principe : répétition d'un grand nombre d'expérience, puis calcul de la moyenne.
- Algorithme peu performant, et impossible à réaliser pour n au dessus de 10 : complexité en $\mathcal{O}(nm^2t)$.

- Analyse déterministe :

- Principe : On calcule directement les matrices itérées P^k .
- Impossible en pratique ($(52!)^2 \approx 7 \times 10^{135}$).
- Possibilité de réduction des P^k en taille n.

Le phénomène de *cut-off*

Questions de programmation

- Analyse fréquentielle :

- Principe : répétition d'un grand nombre d'expérience, puis calcul de la moyenne.
- Algorithme peu performant, et impossible à réaliser pour n au dessus de 10 : complexité en $\mathcal{O}(nm^2t)$.

- Analyse déterministe :

- Principe : On calcule directement les matrices itérées P^k .
- Impossible en pratique ($(52!)^2 \approx 7 \times 10^{135}$).
- Possibilité de réduction des P^k en taille n.

Le phénomène de *cut-off*

Questions de programmation

- Analyse fréquentielle :

- Principe : répétition d'un grand nombre d'expérience, puis calcul de la moyenne.
- Algorithme peu performant, et impossible à réaliser pour n au dessus de 10 : complexité en $\mathcal{O}(nm^2t)$.

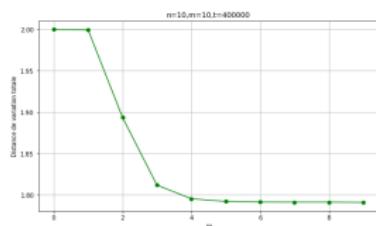
- Analyse déterministe :

- Principe : On calcule directement les matrices itérées P^k .
- Impossible en pratique ($(52!)^2 \approx 7 \times 10^{135}$).
- Possibilité de réduction des P^k en taille n.

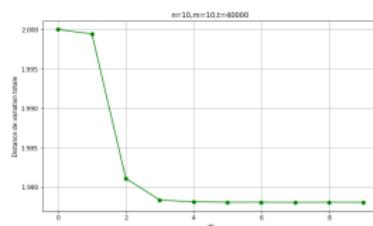
Le phénomène de *cut-off*

Graphes avec l'analyse fréquentielle

$$t = 4 \times 10^4$$



$$t = 4 \times 10^5$$



$$t = 4 \times 10^6$$

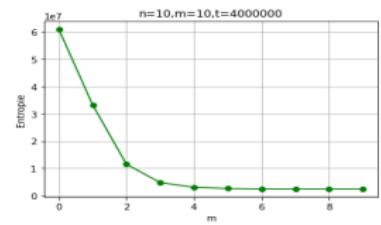
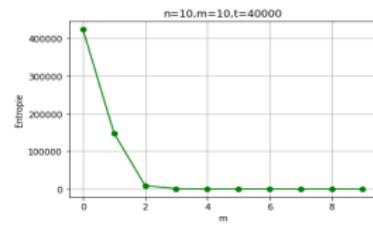
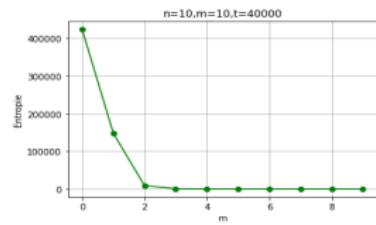
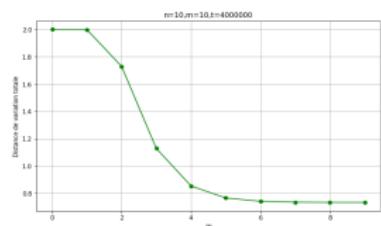


Table – Pour $(n, m) = (10, 10)$

Le phénomène de *cut-off*

Un problème de réduction

- Importance du nombres de *suites croissantes* $r(\sigma)$. On montre que :

$$\forall \sigma \in \mathfrak{S}_n, \mathbb{P}(S_i = \sigma) = \binom{2^i + n - r(\sigma)}{n} \frac{1}{2^{in}}.$$

- Le problème se rapporte alors au nombre de suites croissantes d'une permutation (au plus n).
- Le coefficient général de la matrice de transition dans cette espace est alors :

$$p_{i,j} = \frac{1}{2^n} \binom{n+1}{2i-j} \frac{A_{n,j}}{A_{n,i}},$$

où les $A_{n,i}$ sont les *nombres eulériens*.

Le phénomène de *cut-off*

Un problème de réduction

- Importance du nombres de *suites croissantes* $r(\sigma)$. On montre que :

$$\forall \sigma \in \mathfrak{S}_n, \mathbb{P}(S_i = \sigma) = \binom{2^i + n - r(\sigma)}{n} \frac{1}{2^{in}}.$$

- Le problème se rapporte alors au nombre de suites croissantes d'une permutation (au plus n).
- Le coefficient général de la matrice de transition dans cette espace est alors :

$$p_{i,j} = \frac{1}{2^n} \binom{n+1}{2i-j} \frac{A_{n,j}}{A_{n,i}},$$

où les $A_{n,i}$ sont les *nombres eulériens*.

Le phénomène de *cut-off*

Un problème de réduction

- Importance du nombres de *suites croissantes* $r(\sigma)$. On montre que :

$$\forall \sigma \in \mathfrak{S}_n, \mathbb{P}(S_i = \sigma) = \binom{2^i + n - r(\sigma)}{n} \frac{1}{2^{in}}.$$

- Le problème se rapporte alors au nombre de suites croissantes d'une permutation (au plus n).
- Le coefficient général de la matrice de transition dans cette espace est alors :

$$p_{i,j} = \frac{1}{2^n} \binom{n+1}{2i-j} \frac{A_{n,j}}{A_{n,i}},$$

où les $A_{n,i}$ sont les *nombres eulériens*.

Le phénomène de *cut-off*

Un problème de réduction

On obtient alors un algorithme qui s'exécute en $\mathcal{O}(mn^3)$, et qui fournit les graphes :

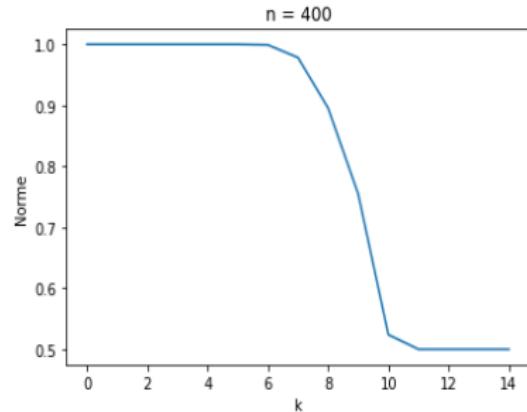
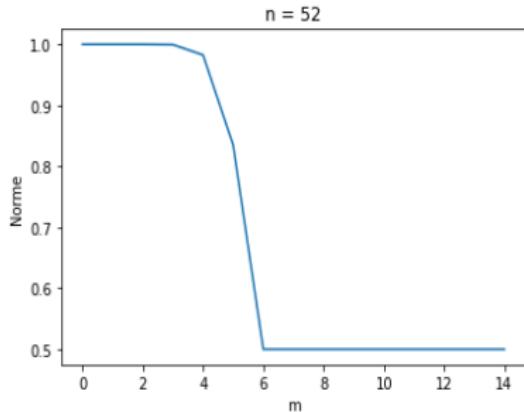


Table – Graphes pour l'algorithme déterministe

Conclusion

- Même résultat qu'en théorie pour $m \sim \frac{3}{2} \log_2(n)$, même pour des valeurs de n qui nous intéressent.
- Possibilité de généralisation à plusieurs paquets : c'est le *a-shuffle*, dans lequel on observe aussi un phénomène de *cut-off* pour $\frac{3}{2\mu} \ln(n)$, où $\mu := \sum_{k=1}^{+\infty} \ln(k)p(k)$.

Conclusion

- Même résultat qu'en théorie pour $m \sim \frac{3}{2} \log_2(n)$, même pour des valeurs de n qui nous intéressent.
- Possibilité de généralisation à plusieurs paquets : c'est le *a-shuffle*, dans lequel on observe aussi un phénomène de *cut-off* pour $\frac{3}{2\mu} \ln(n)$, où $\mu := \sum_{k=1}^{+\infty} \ln(k)p(k)$.

Compléments

Sur le *faro shuffle*

Les isomorphismes sont donnés par le théorème suivant (en particulier, pour $2n = 52$ cartes, on est dans le cadre de (a), où B_n est le *groupe de Weyl*) :

THEOREM. *Let $\langle I, O \rangle$ be the permutation group generated by in and out shuffles of $2n$ cards.*

(a) *If $n \equiv 2 \pmod{4}$ and $n > 6$, then $\langle I, O \rangle$ is isomorphic to B_n and $|\langle I, O \rangle| = n!2^n$. If $n = 6$, then $\langle I, O \rangle$ is a semi-direct product of Z_2^6 with $PGL(2, 5)$.*

(b) *If $n \equiv 1 \pmod{4}$ and $n \geq 5$, then $\langle I, O \rangle$ is the kernel of $\overline{\text{sgn}}$ and $|\langle I, O \rangle| = n!2^{n-1}$.*

(c) *If $n \equiv 3 \pmod{4}$, then $\langle I, O \rangle$ is isomorphic to D_n and $|\langle I, O \rangle| = n!2^{n-1}$.*

(d) *If $n \equiv 0 \pmod{4}$, $n > 12$, and n not a power of 2, then $\langle I, O \rangle$ is the intersection of the kernels of sgn and $\overline{\text{sgn}}$, and $|\langle I, O \rangle| = n!2^{n-2}$. If $2n = 24$, then $\langle I, O \rangle$ is a semi-direct product of Z_2^{11} with the Mathieu group M_{12} of degree 12.*

(e) *If $2n = 2^k$, $\langle I, O \rangle$ is isomorphic to the semi-direct product of Z_2^k by Z_k , where Z_k acts by a cyclic shift and $|\langle I, O \rangle| = k \cdot 2^k$.*

Figure – D'après "The Mathematics of perfect shuffle", P. Diaconis, R.L.Graham, W.M.Kantor.

Démonstrations

Convergence de P^k

En posant $r = r(\sigma)$, on sait que le coefficient général de la matrice P^k est

$$\binom{2^k+n-r}{n} \frac{1}{2^{kn}}. \text{ Montrons alors que } \binom{2^k+n-r}{n} \underset{k \rightarrow +\infty}{\sim} \frac{2^{kn}}{n!}.$$

- D'une part, $\binom{2^k+n-r}{n} = \frac{(2^k+n-r)!}{(2^k-r)!n!}$.

- D'autre, avec la formule de Stirling,

$$\begin{aligned} \frac{(2^k+n-r)!}{(2^k-r)!} &\underset{k \rightarrow +\infty}{\sim} \frac{\sqrt{2\pi}\sqrt{2^k+n-r}}{\sqrt{2\pi}\sqrt{2^k-r}} \frac{(2^k+n-r)^{2^k+n-r}}{(2^k-r)^{2^k-r}} \frac{e^{-(2^k+n-r)}}{e^{-(2^k-r)}} \\ &\underset{k \rightarrow +\infty}{\sim} \sqrt{\frac{2^k+n-r}{2^k-r}} \frac{(2^k+n-r)^{2^k+n-r}}{(2^k-r)^{2^k-r}} e^{-n}. \end{aligned}$$

$$\begin{aligned} \text{Or, } &(2^k + n - r)\ln(2^k + n - r) - (2^k - r)\ln(2^k - r) \\ &= (2^k - r)\ln\left(\frac{2^k+n-r}{2^k-r}\right) + n\ln(2^k + n - r) \\ &= (2^k - r)\ln\left(1 + \frac{n}{2^k-r}\right) + n\ln(2^k + n - r) \\ &= (2^k - r)\left(\frac{n}{2^k-r} + \underset{k \rightarrow +\infty}{o}\left(\frac{1}{2^k}\right)\right) + n\ln(2^k + n - r) \\ &= n + n\ln(2^k + n - r) + \underset{k \rightarrow +\infty}{o}(1). \end{aligned}$$

Démonstrations

Convergence de P^k

$$\begin{aligned} \text{Donc, } \frac{(2^k+n-r)^{2^k+n-r}}{(2^k-r)^{2^k-r}} &= e^{n+n\ln(2^k+n-r)+o_{k \rightarrow +\infty}(1)} \\ &= e^{n+\ln(2^{kn})+o_{k \rightarrow +\infty}(1)}. \end{aligned}$$

$$\text{De plus, } \sqrt{\frac{2^k+n-r}{2^k-r}} = \sqrt{1 + \frac{n}{2^k-r}} = 1 + o_{k \rightarrow +\infty}\left(\frac{1}{2^k}\right).$$

$$\text{Et finalement, } \frac{(2^k+n-r)!}{(2^k-r)!} \underset{k \rightarrow +\infty}{\sim} 2^{kn},$$

de sorte que les coefficients de P^k convergent tous vers $\frac{1}{n!}$ pour $k \rightarrow +\infty$, ce qui achève la démonstration.

Remarques

Sur les mesures de désordre

- Sur l'entropie :
 - Pour tout $\sigma \in \mathfrak{S}_n$, $\log_2(\mathbb{P}(S_k = \sigma)) \leq 0$, d'où le signe '−' pour garantir $H \geq 0$.
 - On adopte la convention $\mathbb{P}(S_k = \sigma)\log_2(\mathbb{P}(S_k = \sigma)) = 0$ lorsque $\mathbb{P}(S_k = \sigma) = 0$.
- Sur la distance totale de variation :
 - Cette mesure est particulièrement intéressante d'après la propriété suivante :

Proposition : Soient X et $(X_k)_{k \in \mathbb{N}}$ des variables aléatoires de loi respectives ν et $(\nu_k)_{k \in \mathbb{N}}$.

$$\text{Alors : } (X_k \xrightarrow{\text{loi}} X) \Leftrightarrow (d_{vt}(\nu_k, \nu) \xrightarrow{k \rightarrow +\infty} 0).$$

Notation et définitions

① Notation de la complexité :

- n est le nombre de cartes totale du paquet (pour notre étude, $n = 52$).
- m est le nombre de *riffle shuffle* exécutés d'affilé (pour $n = 52$, le phénomène de *cut-off* apparaît pour 7 tirages environ, donc $m = 10$).
- t est le nombre de répétition de l'expérience, qui se doit d'être plutôt grand ($t \sim n!$ serait idéal).

Sur les nombres eulériens et les suites croissantes

Définition : On définit les nombres eulériens par récurrence double en posant :

$$\begin{cases} A_{1,1} = 1 \\ A_{1,k} = 0 & \text{si } k > 1 \\ A_{n,k} = kA_{n-1,k} + (r - k + 1)A_{r-1,k-1} & \text{si } n > 1 \end{cases}$$

Définition : Soit $\sigma \in \mathfrak{S}_n$.

On appelle suite croissante de σ toute suite d'entiers consécutifs $(i, i+1, \dots, i+k)$ telle que :

- (maximalité) pour tout $(l, m) \in \mathbb{N}^2 \setminus \{(0, 0)\}$,
 $(i-l, i-l+1, \dots, i+m+k-1, i+m+k)$, n'est pas une suite croissante,
- $\sigma^{-1}(i) < \sigma^{-1}(i+1) < \dots < \sigma^{-1}(i+k)$.

Illustrations

Matrices de biais

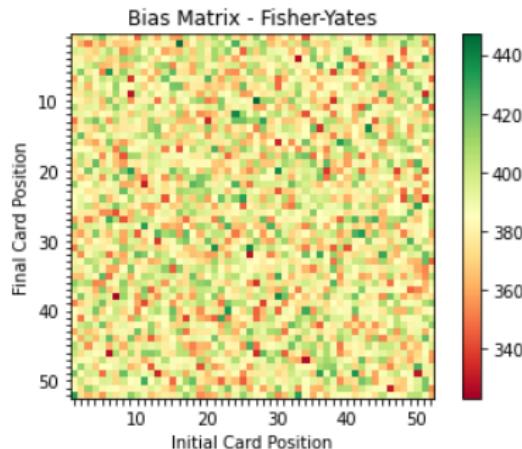
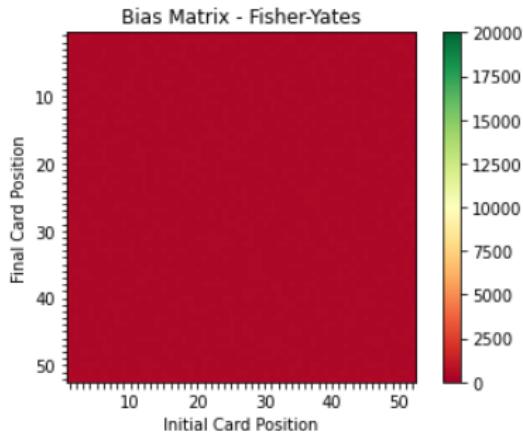


Figure – Matrice de biais du mélange *Fisher-Yates* ...

Figure – ... sur lequel on a zoomé.

Code

Matrice de biais

```
 8 import random
 9 import matplotlib.pyplot as plt
10 import numpy as np
11 import numpy.random as rd
12
13 def riffle_shuffle(cards):
14     n = len(cards)
15     c = rd.binomial(n, 1/2)
16     paquet_gauche = cards[:c]
17     paquet_droit = cards[c:]
18     paquet_final = []
19     for i in range(n):
20         s = len(paquet_gauche) / (n - i)
21         prob = random.random()
22         if prob < s:
23             paquet_final.append(paquet_gauche[0])
24             paquet_gauche.pop(0)
25         else:
26             paquet_final.append(paquet_droit[0])
27             paquet_droit.pop(0)
28     return paquet_final
29
30 def fisher_yates(cards): # melange fisher yates (échange de cartes)
31     n = len(cards)
32     for i in range(n-1, 0, -1):
33         j = rd.randint(i+1)
34         cards[i], cards[j] = cards[j], cards[i]
35     return cards
36
37 def faro_shuffle(cards):
38     n = len(cards) // 2
39     shuffled_cards = []
40     for i in range(n):
41         shuffled_cards.append(cards[i])
42         shuffled_cards.append(cards[i + n])
43     return shuffled_cards
```

Figure – Part1

Code

Matrice de biais

```
45 def generate_bias_matrix(shuffle_func, n, iterations):
46     bias_matrix = np.zeros((n, n), dtype=int)
47     for _ in range(iterations):
48         cards = list(range(n))
49         shuffled_cards = shuffle_func(cards)
50         for i, card in enumerate(shuffled_cards):
51             bias_matrix[i][card] += 1
52     return bias_matrix
53
54 def plot_bias_matrix(bias_matrix, iterations, shuffle_name):
55     n = len(bias_matrix)
56     fig, ax = plt.subplots()
57     im = ax.imshow(bias_matrix, cmap='RdYlGn', vmin=0, vmax=iterations)
58
59     ax.set_xticks(np.arange(n))
60     ax.set_yticks(np.arange(n))
61     ax.set_xticklabels([i if i % 10 == 0 else "" for i in range(1,n+1)])
62     ax.set_yticklabels([i if i % 10 == 0 else "" for i in range(1,n+1)])
63     plt.xlabel('Initial Card Position')
64     plt.ylabel('Final Card Position')
65     plt.title(f'Bias Matrix - {shuffle_name}')
66
67     plt.colorbar(im) # dégradé de couleurs
68     plt.show()
69
70
71 n = 52 # nombre de cartes
72 iterations = 20000 # nombre de mélanges effectués
73
74 bias_matrix_fisher_yates = generate_bias_matrix(fisher_yates, n, iterations)
75 bias_matrix_riffle_shuffle = generate_bias_matrix(riffle_shuffle, n, iterations)
76 bias_matrix_faro = generate_bias_matrix(faro_shuffle, n, iterations)
77
78 plot_bias_matrix(bias_matrix_fisher_yates,iterations,"Fisher-Yates")
79 plot_bias_matrix(bias_matrix_riffle_shuffle,iterations,"Riffle Shuffle")
80 plot_bias_matrix(bias_matrix_faro,iterations,"Faro Shuffle")
```

Figure – Part2

Code

Analyse fréquentielle

```
 8 import math
 9 import matplotlib.pyplot as plt
10 import random
11 import numpy.random as rd
12
13 # Nombre de cartes (n) et nombre de mélanges (t)
14 n = 10
15 t = 4_000_000
16 m = 10
17
18 plt.figure(figsize=(10, 6))
19
20 def fact(n):
21     if n == 0:
22         return 1
23     else:
24         return n * fact(n - 1)
25
26 def riffle_shuffle(cards):
27     n = len(cards)
28     c = rd.binomial(n, 1/2)
29     paquet_gauche = cards[:c]
30     paquet_droit = cards[c:]
31     paquet_final = []
32     for i in range(n):
33         s = len(paquet_gauche) / (n - i)
34         prob = random.random()
35         if prob < s:
36             paquet_final.append(paquet_gauche[0])
37             paquet_gauche.pop(0)
38         else:
39             paquet_final.append(paquet_droit[0])
40             paquet_droit.pop(0)
41     return paquet_final
```

Figure – Part1

Code

Analyse fréquentielle

```
44 def dict_melange(n, m, t):
45     d = {}
46     for _ in range(t):
47         jeu_de_cartes = list(range(n))
48         for _ in range(m):
49             jeu_de_cartes = riffle_shuffle(jeu_de_cartes)
50         jeu = tuple(jeu_de_cartes)
51         if jeu in d:
52             d[jeu] += 1
53         else:
54             d[jeu] = 1
55     return d
56
57 # Fonction pour calculer distance et entropie
58 def distance(n,m,t):
59     d = dict_melange(n,m,t)
60     dist = []
61     f = fact(n)
62     for c in d:
63         dist.append(abs(1/f - (d[c]/t)))
64     return dist
65
66
67 def entropie(n,m,t):
68     d = dict_melange(n,m,t)
69     dist = []
70     for c in d:
71         dist.append(d[c]*math.log(d[c]))
72     return dist
```

Figure – Part2

Code

Analyse fréquentielle

```
75 # Fonctions pour calculer la moyenne des distances
76 def moy_dist(n,m,t):
77     dist = distance(n,m,t)
78     s = sum(dist)
79     s = s
80     return s
81
82 def entropie_moy(n,m,t):
83     dist = entropie(n,m,t)
84     s = sum(dist)
85     s += 1 - (len(dist)/fact(n))
86     return s
87
88 # Fonctions pour calculer la variation totale
89 def variation_totale(n, m, t):
90     var = []
91     for i in range(m):
92         moy = moy_dist(n,i,t)
93         var.append(moy)
94     return var
95
96 def entropie_totale(n,m,t):
97     var = []
98     for i in range(m):
99         moy = entropie_moy(n,i,t)
100        var.append(moy)
101    return var
```

Figure – Part3

Code

Analyse fréquentielle

```
103 valeurs_Dn = variation_totale(n,m,t)
104 print(valeurs_Dn)
105
106 valeurs_m = list(range(len(valeurs_Dn)))
107
108 plt.plot(valeurs_m, valeurs_Dn, marker='o', linestyle='-', color='green', label=f'D{n}(t)')
109
110 # Définir les étiquettes et le titre
111 plt.xlabel('t')
112 plt.ylabel(f'D{n}(t)')
113 plt.title(f'Distance de Variation D{n}(t) avec Phénomène de Coupe')
114
115 # Afficher le graphique
116 plt.grid(True)
117 plt.show()
```

Figure – Part4

Code

Analyse déterministe

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # n est la taille de la matrice, n = 54
5
6 ## Fonctions annexes ##
7
8 def euler_numbers(n):
9     # Renvoie un tableau [A(n,1),A(n,2),...,A(n,n)] des nombres eulériens de rang n.
10    tab = [1 if i == 0 else 0 for i in range(n)]
11
12    for i in range(1,n):
13        tab_aux = [1 for _ in range(n)]
14        for j in range(1,i):
15            tab_aux[j] = (j+1)*tab[j] + (i-j+1)*tab[j-1]
16        tab = tab_aux
17
18    return tab
19
20
21 def coeff_binom(n):
22     # Renvoie un tableau [(0 n),(1 n),...,(n n)] des coefficients binomiaux de rang n.
23     tab = [1 if i == 0 else 0 for i in range(n)]
24
25    for i in range(1,n+1):
26        tab_aux = [1 for _ in range(n+1)]
27        for j in range(1,i):
28            tab_aux[j] = tab[j] + tab[j-1]
29        tab = tab_aux
30
31    return tab
32
33 def fact(n):
34     # Renvoie n!
35     if n == 0:
36         return 1
37     else:
38         return n * fact(n-1)
```

Figure – Part1

Code

Analyse déterministe

```
40 def norme_un_matrice(M):
41     # Renvoie la norme un (maximum de la somme des modules sur les lignes) d'une matrice.
42     norme = 0
43     n = len(M)
44
45     for i in range(n):
46         a = 0
47         for j in range(n):
48             a += abs(M[i][j])
49             if norme < a:
50                 norme = a
51     return norme
52
53 ## Corps du code ##
54
55 def constr_P(n):
56     # Renvoie la matrice P n*n de terme général donné dans la littérature
57     a = coeff_binom(n)
58     b = euler_numbers(n)
59     return [[(a[2*i-j+1]*b[j]) / (b[i]**(2**n)) if (0 <= 2*i-j+1 and 2*i-j+1 <= n)
60             else 0 for j in range(n)] for i in range(n)]
61
62
63 def constr_P_inf(n):
64     # Renvoie la matrice P^{inf} n*n de terme général donné dans la littérature
65     sigma = euler_numbers(n)
66     f = fact(n)
67     return [[sigma[j]/f for j in range(n)] for i in range(n)]
```

Figure – Part2

Code

Analyse déterministe

```
40 def norme_un_matrice(M):
41     # Renvoie la norme un (maximum de la somme des modules sur les lignes) d'une matrice.
42     norme = 0
43     n = len(M)
44
45     for i in range(n):
46         a = 0
47         for j in range(n):
48             a += abs(M[i][j])
49             if norme < a:
50                 norme = a
51
52     return norme
53
54 ## Corps du code ##
55
56 def constr_P(n):
57     # Renvoie la matrice P n*n de terme général donné dans la littérature
58     a = coeff_binom(n)
59     b = euler_numbers(n)
60     return [[(a[2*i-j+1]*b[j]) / (b[i]**(2**n)) if (0 <= 2*i-j+1 and 2*i-j+1 <= n)
61             else 0 for j in range(n)] for i in range(n)]
62
63 def constr_P_inf(n):
64     # Renvoie la matrice P^{inf} n*n de terme général donné dans la littérature
65     sigma = euler_numbers(n)
66     f = fact(n)
67     return [[sigma[j]/f for j in range(n)] for i in range(n)]
```

Figure – Part3