Chapitre 2

Algorithmes probabilistes

Hugo Salou MPI*

Dernière mise à jour le 13 mai 2023

Table des matières

1 Introduction

Dans ce chapitre, on s'intéresse aux algorithmes probabilistes de deux types : Monte-Carlo et Las-Vegas. L'idée est de donner une définition plus mathématique d'un « algorithme probabiliste » et de l'influence de l'aléatoire.

Définition : Un algorithme *déterministe* est un algorithme tel que pour chaque entrée I de l'algorithme, l'exécution de l'algorithme produit toujours exactement la même suite d'états.

Remarque :
Un algorithme déterministe produit donc toujours les même sorties sur les mêmes entrées.

Définition : Un algorithme *probabiliste* est un algorithme opérant sur un ensemble $\mathscr{E}$, tel que la suite d'états obtenus par exécution de l'algorithme sur une entrée $e \in \mathscr{E}$ est une variable aléatoire.

Remarque :
Avec cette définition, un algorithme déterministe est un algorithme probabiliste.

Exemple :
On considère le problème suivant :

$$\text{Problème Tri} : \begin{cases} \text{Entrée} & : \text{un tableau } T \text{ de taille } n \\ \text{Sortie} & : T \text{ trié.} \end{cases}$$

Une réponse à ce problème est l'algorithme nommé Bozosort décrit ci-dessous. On le nomme aussi « tri aléatoire. »

Algorithme 1 Bozosort

Entrée T un tableau
1 : **tant que** T non trié **faire**
2 : $\quad i \leftarrow \mathcal{U}(\llbracket 1, n-1 \rrbracket)$
3 : $\quad j \leftarrow \mathcal{U}(\llbracket 1, n-1 \rrbracket)$
4 : $\quad$ Échanger i et j dans le tableau T

On étudie l'algorithme ci-dessus : il est trivialement partiellement correct (i.e. s'il est correct). En effet, par négation de la condition de boucle, on a T trié.

Le temps d'exécution de l'algorithme est difficile à estimer. L'algorithme peut ne pas terminer.

Exemple :
On considère à présent le problème ci-dessous : approximer π. L'algorithme tire des points au hasard dans un carré unité, et regarde si le point est dans le disque unité.

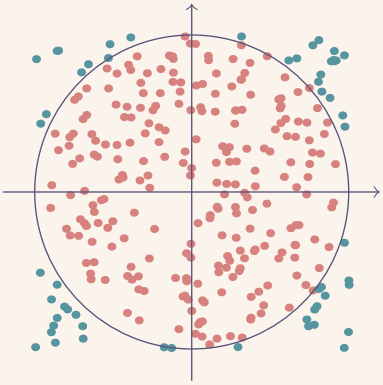

Figure 1 – Algorithme de Monte-Carlo pour approximer π

On compte le nombre de points dans le disque, et ceux en dehors. Avec un grand nombre de points, on approxime le ratio de l'aire du disque et de l'aire du carré. Puis, on calcule

$$4 \times \left(\frac{\#\bullet}{\#\bullet + \#\bullet} \right) \approx \pi.$$

Le temps d'exécution dépend uniquement des paramètres de précision de l'algorithme, pas des tirages. Par contre, la qualité de la réponse dépend des choix aléatoires.

Définition (Algorithme de type Las Vegas) **:** Étant donné un problème P, un algorithme probabiliste répondant au problème P est dit de type Las Vegas dès lors que, s'il se termine, c'est en donnant une réponse correcte.

Définition (Algorithme de type Monte-Carlo) **:** Étant donné un problème P, un algorithme probabiliste répondant au problème P est dit de type Monte-Carlo dès lors que son temps d'exécution dépend uniquement de son entrée. L'algorithme peut cependant répondre de manière erronée au problème P avec une « certaine » probabilité.

Remarque :
Dans le cas d'un problème de décision (la réponse de l'algorithme est oui ou non), un algorithme de type Monte-Carlo est dit

- « à erreur unilatérale » s'il existe une des réponses (oui ou non) r telle que, si l'algorithme répond r, alors il a raison (r est la réponse au problème) ;
- « à erreur bilatérale » si pour chaque réponse l'algorithme se trompe avec une probabilité non nulle.

Exemple :
On considère le problème : étant donné un tableau $T \in \{0,1\}^n$ tel que T contient p fois la valeur '0', avec $0 < p < n$, on cherche si, pour $i \in [\![1, n-1]\!]$, $T[i] = 1$.

Une réponse à ce problème est un algorithme de type Monte-Carlo, comme celui ci-dessous.

Algorithme 2 Algorithme de Monte-Carlo pour répondre au problème

Entrée $k \in \mathbb{N}$ et T un tableau
1 : $i \leftarrow 0$
2 : **pour** $j \in [\![1, k]\!]$ **faire**
3 : $\quad i \leftarrow \mathcal{U}([\![1, n-1]\!])$
4 : $\quad$ **si** $T[i] = 1$ **alors**
5 : $\quad\quad$ **retourner** i
6 : **retourner** i

Mais, on peut également donner un algorithme de Las-Vegas répondant aussi au même problème.

Algorithme 3 Algorithme de Las-Vegas pour répondre au problème

Entrée T un tableau
1 : $i \leftarrow \mathcal{U}([\![1, n-1]\!])$
2 : **tant que** $T[i] \neq 1$ **faire**
3 : $\quad i \leftarrow \mathcal{U}([\![0, n-1]\!])$
4 : **retourner** i

Étudions l'algorithme de Las-Vegas : la correction partielle est validée. Étudions la terminaison : fixons T un tableau de taille n contenant p occurrences de '0' avec $0 < p < n$. Notons $(X_\ell)_{\ell \in \mathcal{D}}$ la suite des variables aléatoires donnant la valeur produite par le ℓième appel à $\mathcal{U}([\![0, 1]\!])$. Remarquons que $\mathcal{D}$ est une variable aléatoire : en effet c'est $[\![0, N]\!]$ pour un certain $N \in \mathbb{N}$ si l'algorithme se termine ; sinon, on a $\mathcal{D} = \mathbb{N}$. Notons A_q l'événement « l'algorithme s'arrête après q appels au générateur $\mathcal{U}([\![0, n-1]\!])$. On a donc

$$A_q = \text{“}T[X_0] = 0 \wedge T[X_1] = 0 \wedge \cdots \wedge T[X_{q-2}] = 0 \wedge T[X_{q-1}] = 1\text{”}.$$

D'où en passant aux probabilités, on a

$$P(A_q) = P(\text{“}T[X_0] = 0 \wedge T[X_1] = 0 \wedge \cdots \wedge T[X_{q-2}] = 0 \wedge T[X_{q-1}] = 1\text{”})$$

et, par indépendance, on a donc

$$P(A_q) = P(T[X_{q-1} = 1) \times \Big(\prod_{j=0}^{q-2} P(T[X_j] = 0) \Big).$$

Or, $\forall j \in [\![0, q-2]\!]$, $P(T[X_j] = 0) = \frac{p}{n}$ par uniformité, et, de plus, $P(T[X_{q-1}] = 1) = \frac{n-p}{n}$ par uniformité. On pose $\rho = \frac{p}{n}$, et donc $P(A_q) = \rho^{q-1}(1-\rho)$. Notons N l'événement « l'algorithme ne se termine pas » et calculons

$$\begin{aligned}
P(N) &= 1 - P(\bar{N}) \\
&= 1 - P\Big(\bigvee_{q \in \mathbb{N}^\star} A_q \Big) \\
&= 1 - \sum_{q \in \mathbb{N}^\star} P(A_q) \\
&= 1 - \sum_{q \in \mathbb{N}^\star} \rho^{q-1}(1-\rho) \\
&= 1 - \frac{1-\rho}{1-\rho} \\
&= 0
\end{aligned}$$

Soit $\mathcal{T}$ la variable aléatoire indiquant le temps d'arrêt de l'algorithme (en nombre d'itérations). On calcule l'espérance de $\mathcal{T}$:

$$\begin{aligned}
\mathrm{E}(\mathcal{T}) &= \sum_{t=1}^{+\infty} t \times P(\mathcal{T} = t) \\
&= \sum_{t=1}^{+\infty} t \times \rho^{t-1}(1-\rho) \\
&= (1-\rho) \sum_{t=1}^{+\infty} t \times \rho^{t-1} \\
&= (1-\rho) \sum_{t=1}^{+\infty} \sum_{k=0}^{t-1} \rho^t \\
&= (1-\rho) \sum_{k=0}^{+\infty} \sum_{t=k+1}^{+\infty} \rho^{t-1} \\
&= \dots\dots \text{ [1]} \\
&= \frac{1}{1-\rho}
\end{aligned}$$

Étudions maintenant l'algorithme de Monte-Carlo. Il se termine trivialement, et la probabilité d'erreur est $\underbrace{\rho \times \cdots \times \rho}_{k}$. Par exemple, pour $\rho = \frac{1}{2}$, et $k = 80$, la probabilité d'erreur est de $\frac{1}{2^{80}}$.

2 Algorithme de Monte-Carlo

On considère le problème : « étant donné trois matrices A, B, C de $\mathcal{M}_n(\mathbb{Z}/2\mathbb{Z})$, a-t-on $A \cdot B = C$? »

Un algorithme trivial serait de calculer $A \cdot B$ et on vérifie, point à point, que $A \cdot B = C$. La complexité cet algorithme est en $\Theta(n^3)$ à cause du produit matriciel.

Un algorithme de Monte-Carlo serait le suivant.

Algorithme 4 Algorithme de Monte-Carlo répondant au problème

Entrée A, B, C trois matrices et $k \in \mathbb{N}$

```
pour j ∈ ⟦1, k⟧ faire
    r ← 𝒰((ℤ/2ℤ)^n)        ▷ n
    r_1 ← B · r            ▷ n^2
    r_2 ← A · r_1          ▷ n^2
    r_3 ← C · r            ▷ n^2
    si r_3 ≠ r_2 alors
        retourner Non
retourner Oui
```

Dans le pire cas, la complexité est en $k \times n^2$. On cherche la probabilité d'erreur de cet algorithme. Pour cela, on utilise le lemme suivant.

1. à faire

Lemme : Si $D \neq 0$, et $r \sim \mathcal{U}\big((\mathbb{Z}/2\mathbb{Z})^n\big)$, alors $P(D \cdot r = 0) \leqslant \frac{1}{2}$.

Preuve :
Si $D \in \mathcal{M}_n(\mathbb{Z}/2\mathbb{Z}) \setminus \{0\}$, alors il existe i et j tels que $D_{i,j} \neq 0$. Si $D \cdot r = 0$, on a

$$\sum_{k=1}^{n} D_{i,k} r_k = 0 \quad \text{et donc} \quad r_k = -\sum_{\substack{k=1 \\ k \neq j}}^{n} D_{i,k} r_k.$$

Donc, si $r_j \neq \sum_{\substack{k=1 \\ k \neq j}}^{n} D_{ik} r_k$, alors $P(D \cdot r \neq 0) \geqslant P\Big(r_j \neq \sum_{\substack{k=1 \\ k \neq j}}^{n} D_{i,k} r_k\Big)$. On note E_0 l'événement « $r_j = 0$ et $\sum_{\substack{k=1 \\ j \neq j}}^{n} D_{i,k} r_k = 1$ » et E_1 l'événement « $r_j = 1$ et $\sum_{\substack{k=1 \\ j \neq j}}^{n} D_{i,k} r_k = 0$. » D'où

$$P\Big(r_j \neq \sum_{\substack{k=1 \\ j \neq j}}^{n} D_{i,k} r_k\Big) = P(E_0 \vee E_1).$$

Par incompatibilité, on a $P(E_0 \vee E_1) = P(E_0) + P(E_1)$, d'où

$$\forall a \in \{0,1\}, \quad P(E_a) = P\Big(r_j = a \wedge \sum_{\substack{k=1 \\ j \neq j}}^{n} D_{i,k} r_k = 1 - a\Big)$$

et, par indépendance,

$$\forall a \in \{0,1\} \quad P(E_a) = P(r_j = a) \cdot P\Big(\sum_{\substack{k=1 \\ j \neq j}}^{n} D_{i,k} r_k = 1 - a\Big) = \frac{1}{2} P(\dots).$$

D'où

$$P\Big(r_j \neq \sum_{\substack{k=1 \\ j \neq j}}^{n} D_{i,k} r_k\Big) = \frac{1}{2}\Big[P\Big(\sum_{\substack{k=1 \\ j \neq j}}^{n} D_{i,k} r_k = 1\Big) + P\Big(\sum_{\substack{k=1 \\ j \neq j}}^{n} D_{i,k} r_k = 0\Big)\Big]$$

et, par incompatibilité,

$$P\Big(r_j \neq \sum_{\substack{k=1 \\ j \neq j}}^{n} D_{i,k} r_k\Big) = \frac{1}{2}\frac{1}{2} P\Big(\sum_{\substack{k=1 \\ j \neq j}}^{n} D_{i,k} r_k \in \{0,1\}\Big) = \frac{1}{2}.$$

□

D'où, l'algorithme ci-dessous est tel que sa probabilité d'échec est de $\frac{1}{2^k}$. Or, l'algorithme a une complexité de $\mathcal{O}(k\,n^2)$.

3 Algorithme de type Las-Vegas

On étudie le tri rapide. On considère les fonctions “Partitionner,” puis “Tri Rapide.”

Algorithme 5 Fonction “Partitionner” utilisée dans le tri rapide

Entrée T le tableau à trier, g, d et p trois entiers (bornes du tableau)
Sortie un entier J et le sous-tableau $T[g..d]$ est modifié en $\bar{T}$ de sorte que $\bar{T}$[2] $[J] = \underline{T}[p]$, et $\forall i \in [\![g, J-1]\!], \bar{T}[i] \leqslant \bar{T}[J]$, et $\forall i \in [\![J+1, d]\!], \bar{T}[i] \geqslant \bar{T}[J]$, et $\forall i \in [\![0, n-1]\!] \setminus [\![g, d]\!]$, $\bar{T}[i] = \underline{T}[i]$, et $\bar{T}$ est une permutation de $\underline{T}$.

```
1 : ÉCHANGER(T, J, d)
2 : J ← g
3 : I ← g
4 : tant que I < d faire
5 :     si T[I] > T[d] alors        ▷ Cas "T[I] > pivot"
6 :         I ← I + 1
7 :     sinon                       ▷ Cas "T[I] ⩽ pivot"
8 :         ÉCHANGER(T, I, J)
9 :         J ← J + 1
10 :        I ← I + 1
11 : ÉCHANGER(T, J, d)
12 : retourner J
```

Remarque :
On admet que $\bar{T}$ est une permutation de $\underline{T}$. On admet également que, $\forall i \in [\![0, n-1]\!] \setminus [\![g, d]\!]$, $\bar{T}[i] = \underline{T}[i]$.

Lemme : “Partitionner” est correct.

Preuve :
On considère

$$(\mathcal{I}) : \begin{cases} \forall k \in [\![g, I]\!],\ T[k] \leqslant T[d] & (1) \\ \forall k \in [\![J, I-1]\!],\ T[k] > T[d] & (2) \\ g \leqslant J \leqslant I \leqslant d & (3) \end{cases}.$$

— Montrons que $\mathcal{I}$ est vrai initialement : à l'initialisation, $I = g$ et $J = g$, donc les trois propriétés $(\mathcal{I})$ sont trivialement vraies.
— Montrons que l'invariant $(\mathcal{I})$ se propage. Notons $\underline{I}$, $\underline{J}$, et $\underline{T}$ les valeurs de I, J et T avant itération de boucle. Notons également $\bar{I}$, $\bar{J}$ et $\bar{T}$ les valeurs de I, J et T après cette même itération de boucle. Supposons que $\underline{I}$, $\underline{J}$ et $\underline{T}$ vérifient $(\mathcal{I})$, et la condition de boucle. Montrons que $\bar{I}$, $\bar{J}$ et $\bar{T}$ vérifient $(\mathcal{I})$. On a donc, d'après $(\mathcal{I})$,

$$\begin{cases} \forall k \in [\![g, \underline{J}-1]\!],\ \underline{T}[k] \leqslant \underline{T}[d] \\ \forall k \in [\![\underline{T}, \underline{I}-1]\!],\ \underline{T}[k] > \underline{T}[d] \\ g \leqslant \underline{J} \leqslant \underline{I} \leqslant d. \end{cases}$$

Mais aussi, d'après la condition de boucle, $\underline{I} \leqslant d$. Mais également, d'après le programme,
  — si $\underline{T}[\underline{I}] > \underline{T}[d]$, alors $\bar{I} = \underline{I} + 1$, $\bar{T} = \underline{T}$, et $\bar{J} = \underline{J}$;
  — sinon si $\underline{T}[\underline{I}] \leqslant \underline{T}[d]$, et donc $\bar{J} = \underline{J} + 1$, $\bar{I} = \underline{I} + 1$, $\forall k \in [\![0n, -1]\!] \setminus \{\underline{I}, \underline{J}\}$, $\bar{T}[k] = \underline{T}[k]$, et $\bar{T}[\bar{I}] = \underline{T}[\underline{J}]$, et $\bar{T}[\underline{J}] = \underline{T}[\underline{I}]$.

Cas 1 $\underline{T}[\underline{I}] > \underline{T}[d]$, alors
  (3) $g \leqslant \underline{J} = \bar{J} \leqslant \underline{I} < \bar{I} \leqslant d$.
  (1) Soit $k \in [\![g, \bar{J}-1]\!]$, on a donc $k \in [\![g, \underline{J}-1]\!]$, et donc $\bar{T}[k] = \underline{T}[k] \leqslant \underline{T}[d] = \bar{T}[d]$.
  (2) Soit $k \in [\![\bar{J}, \bar{I}-1]\!]$,
    — si $k \in [\![\underline{J}, \underline{I}-1]\!]$, alors $\bar{T}[k]$ă $= \underline{T}[k] > \underline{T}[d] = \bar{T}[d]$.
    — si $k = \bar{I} - 1 = \underline{I}$, par condition `if`, alors $\bar{T}[\underline{I}] = \underline{T}[\underline{I}] > \underline{T}[d] = \bar{T}[d]$.
Cas 2 $\underline{T}[\underline{I}] \leqslant \underline{T}[d]$
  (3) On a $\underline{J} \leqslant \underline{I}$, donc $\underline{J} + 1 \leqslant \underline{I} + 1$, d'où $g \leqslant \underline{J} + 1 = \bar{J} \leqslant \bar{I} \leqslant d$.
  (1) Soit $k \in [\![g, \bar{J}-1]\!]$, donc
    — si $k \in [\![g, \underline{J}-1]\!]$, alors $\bar{T}[k] = \underline{T}[k] \leqslant \underline{T}[d] = \bar{T}[d]$.

2. La notation $\bar{T}$ représente le tableau T après l'algorithme, et la notation $\underline{T}$ représente le tableau T avant l'algorithme.

— si $k = \bar{J} - 1 = \underline{J}$, alors $\bar{T}[k] = \bar{T}[\underline{J}] = \underline{T}[\underline{I}] \leqslant \underline{T}[d] = \bar{T}[d]$.

(2) Soit $k \in [\![\bar{J}, \bar{I} - 1]\!]$, alors

— si $k \in [\![\bar{J}, \underline{J} - 1]\!]$, alors, comme $\bar{J} \geqslant \underline{J}$, et donc $\bar{T}[k] = \underline{T}[k] > \underline{T}[d] = \bar{T}[d]$.

— si $k = \bar{I} - 1 = \underline{I}$, et donc $\bar{T}[k] = \bar{T}[\underline{I}] = \bar{T}[\underline{J}] > \underline{T}[d] = \bar{T}[d]$.

Ainsi, $(\mathcal{I})$ est un invariant, et donc, en sortie de boucle, I, J et T sont tels que

$$\left.\begin{array}{l} \forall k \in [\![g, J-1]\!]\,,\ T[k] \leqslant T[d] \\ \forall k \in [\![J, I-1]\!]\,,\ T[k] > T[d] \\ g \leqslant J \leqslant I \leqslant d \end{array}\right\} \quad \text{et} \quad I \geqslant d,$$

la négation de la condition de boucle. On a donc $I = d$, et donc en fin de programme,

$$\forall k \in [\![g, J-1]\!]\,,\ T[k] \leqslant T[J] \quad \text{et} \quad \forall k \in [\![J+1, I]\!]\,,\ T[k] > T[J].$$

□

Algorithme 6 Tri rapide

Entrée T un tableau, g et d les bornes de ce tableau

```
1 : si d > g alors
2 :     p ← ChoixPivot(T, g, d)
3 :     J ← Partition(T, g, d, p)
4 :     TriRapide(T, g, J − 1)
5 :     TriRapide(T, J + 1, d)
```

La fonction "Tri(T)" est donc définie comme TriRapide$(T, 0, n-1)$ si T est un tableau de taille n.

Étudions rapidement l'influence du choix du pivot.

Cas 1 On définit "ChoixPivot$(T, g, d) = g$." Ainsi

À faire : Figure

Figure 2 – Arbre des appels récursifs de "TriRapide" avec le pivot à gauche

Ainsi, la complexité de cet algorithme, avec ce choix de pivot, est en $(n-1) + (n-2) + (n-3) + \cdots + 2 = \Theta(n^2)$.

Cas 2 On définit maintenant le choix du pivot comme l'indice de la médiane.

À faire : Figure

Figure 3 – Arbre des appels récursifs de "TriRapide" avec le pivot à la médiane

Rédigeons-le rigoureusement : soit $C_n = \max_{T \text{ tableau de taille } n} C(T)$. Posons $(u_p)_{p\in\mathbb{N}} = (C_{2^p})_{p\in\mathbb{N}}$. D'après l'algorithme de "TriRapide," on a

$$\begin{aligned} u_{p+1} &= 2^{p+1} - 1 + u_p + u_p \\ &= 2^{p+1} - 1 + 2u_p \\ &= (2^{p+1} - 1) + 2(2^p - 1) + 2^2 u_{p-1} \\ &= 2^{p+1} - 1 + 2^{p+1} - 2 + 2^2 u_{p-1} \\ &= 2^{p+1} - 1 + 2^{p+1} - 2 + 2^2(2^{p-1} - 1 + 2u_{p-2}) \\ &= 2^{p+1} - 1 + 2^{p+1} - 2 + 2^{p+1} - 2^2 + 2^3 u_{p-2}. \end{aligned}$$

On a donc $u_0 = 1$ et $u_p = p \times 2^p - (2^p - 1)$. Or, la suite $(c_n)_{n\in\mathbb{N}}$ est croissante. Or,

$$\forall n \in \mathbb{N},\ 2^{\lfloor \log_2 n \rfloor} \leqslant n \leqslant 2^{\lfloor \log_2 n \rfloor + 1}$$

donc

$$u_{\lfloor \log_2 n \rfloor} \leqslant C_n \leqslant u_{\lfloor \log_2 n \rfloor + 1}.$$

D'où

$$c_n \leqslant (\lfloor \log_2 n \rfloor + 1) \times 2^{\lfloor \log_2 n \rfloor + 1} - 2^{\lfloor \log_2 n \rfloor - 1}.$$

Et donc, on en déduit que $c_n = \Theta\big(n \log_2(n)\big)$.

Remarque (Notations) :
On fixe un tableau T de taille n. De plus, on suppose dans toute la suite, que $T \in \mathfrak{S}_n$.[a] On note alors $X_g^d[T]$ la variable aléatoire indiquant le nombres de comparaisons effectuées par l'algorithme TriRapide(T, g, d), dès lors que $T(\llbracket g, d \rrbracket) \subseteq \llbracket g, d \rrbracket$.

On note de plus, $\mathrm{E}\big[X_g^d[T]\big]$ l'espérance de cette variable aléatoire.

a. T est une permutation de n éléments. Ici, $\mathfrak{S}_n$ représente l'ensemble des permutations de $\llbracket 1, n \rrbracket$.

Théorème : Le nombre moyen de comparaisons effectuées par l'algorithme de tri rapide pour une entrée T de taille n est équivalent à $2n \ln n$. Autrement dit,

$$\mathrm{E}\big[X_0^{n-1}[T]\big] \sim 2n \ln n.$$

Preuve :
— Lorsque $d \leqslant g$, alors $X_g^d[T] = 0$.
— Lorsque $g < d$,
— dans l'éventualité d'un choix de pivot d'indice $p \in \llbracket g, d \rrbracket$, le nombre de comparaisons est alors

$$\underbrace{d - g - 1}_{\text{coût de Partition}(T,\, g,\, d,\, p)} + \overbrace{X_g^{T[p]-1}\Big[T^{g,d,p}\Big]}^{\text{coût du } 1^{\text{er}} \text{ appel récursif}} + \underbrace{X_{T[p]+1}^{d}\Big[T^{g,d,p}\Big]}_{\text{coût du } 2^{\text{nd}} \text{ appel récursif}}$$

où $T^{g,d,p}$ est le tableau T après appel à Partition(T, g, d, p). D'où

$$\begin{aligned}
\mathrm{E}\left[X_g^d[T]\right] &= \sum_{j=g}^{d} \mathrm{E}\Big[X_g^d[T]\Big|_{p=j}\Big] \cdot P(p = j) \\
&= \frac{1}{d-g+1} \sum_{j=g}^{d} \mathrm{E}\Big[X_g^d[T]\Big|_{p=j}\Big] \\
&= \frac{1}{d-g+1} \sum_{j=g}^{d} \Big(d - g - 1 + \mathrm{E}\Big[X_g^{T[j]-1}\big[T^{g,d,j}\big]\Big] \\
&\qquad\qquad + \mathrm{E}\Big[X_{T[j]+1}^{d}\Big[T^{g,d,j}\Big]\Big]\Big) \\
&= (d - g - 1) + \frac{1}{d-g+1} \sum_{k=g}^{d} \Big(\mathrm{E}\Big[X_g^{k-1}\Big[T^{g,d,T^{-1}[k]}\Big]\Big] \\
&\qquad\qquad + \mathrm{E}\Big[X_{k+1}^{d}\Big[T^{g,d,T^{-1}[k]}\Big]\Big]\Big)
\end{aligned}$$

car $T : \llbracket g, d \rrbracket \to \llbracket g, d \rrbracket$ est une bijection.

Soit donc la suite $(c_\ell)_{\ell \in \mathbb{Z}}$ définie par

$$\begin{cases} \forall \ell \in \mathbb{Z}^-, & c_\ell = 0 \\ \forall \ell \in \mathbb{N}^\star, & c_\ell = \mathrm{E}\big[X_0^\ell[\mathrm{id}]\big]. \end{cases}$$

On a alors

$$\forall \ell \in \mathbb{N}^\star, \quad c_\ell = (\ell-1) + \frac{1}{\ell-1}\sum_{k=0}^{\ell}(c_{k-1} - c_{\ell-k-1})$$

$$c_\ell = (\ell-1) + \frac{2}{\ell+1}\sum_{k=1}^{\ell-1} c_k$$

$$(\ell+1)c_\ell = (\ell+1)(\ell-1) + 2\sum_{k=1}^{\ell-1} c_k$$

D'où, $\ell c_\ell = \ell(\ell-2) + 2\sum_{k=1}^{\ell-2} c_k$, et donc

$$(\ell+1)c_\ell - \ell c_{\ell-1} = \ell^2 - 1 - \ell^2 + 2\ell + 2c_{\ell-1}.$$

On en déduit donc que

$$(\ell+1)c_\ell - (\ell+2)c_{\ell-1} = 2\ell - 1$$

et donc

$$\frac{c_\ell}{\ell+2} - \frac{c_{\ell-1}}{\ell+1} = \frac{2\ell-1}{(\ell+1)(\ell+2)}.$$

Soit alors $(u_\ell)_{\ell\in\mathbb{N}} = \big(c_\ell/(\ell+2)\big)_{\ell\in\mathbb{N}}$, et $u_0 = 0$. Alors

$$u_\ell = \sum_{k=1}^{n}(u_k - u_{k-1}) = \sum_{k=1}^{n}\frac{2k-1}{(k+1)(k+2)}$$

or $\frac{2k-1}{(k+1)(k+2)} \sim \frac{2}{k}$, et $\sum_{k\geqslant 1}\frac{2}{k}$ diverge donc $u_\ell \sim \sum_{k=1}^{\ell}\frac{2}{k} \sim 2\ln\ell$. On en déduit donc que $c_\ell \sim 2\ell\ln\ell$.

□

Dans la preuve précédente, on a utilisé le lemme suivant.

Lemme : Soit $(g,d) \in \mathbb{N}^2$ et soit $T \in \mathfrak{S}_n$ une permutation telle que $T(\llbracket g,d \rrbracket) \subseteq \llbracket g,d \rrbracket$.

$$\mathrm{E}\Big[X_g^d[T]\Big] = \mathrm{E}\Big[X_0^{d-g}[\mathrm{id}]\Big].$$

Preuve (par récurrence forte sur $d-g=\ell \in \mathbb{N}$) :
— Soient $(g,d) \in \mathbb{N}^2$ tel que $d-g=0$. Soit $T \in \mathfrak{S}_n$ telle que $T(\llbracket g,d \rrbracket) \subseteq \llbracket g,d \rrbracket$. On a bien $X_g^d[T] = 0 = X_0^{d-g}[\mathrm{id}]$.
— On remarque, par hypothèse de récurrence,

$$\mathrm{E}\Big[X_g^{k-1}\Big[\overbrace{T^{g,d,T^{-1}[k]}}^{k-1-g<d-g}\Big]\Big] = \mathrm{E}\Big[X_0^{k-1-g}[\mathrm{id}]\Big]$$

et

$$\mathrm{E}\Big[X_{k-1}^{d}\Big[T^{g,d,T^{-1}[k]}\Big]\Big] = \mathrm{E}\Big[X_0^{d-k-1}[\mathrm{id}]\Big].$$

On a alors

$$\begin{aligned}
&\mathrm{E}\Big[X_g^d[T]\Big] \\
&= (d-g-1) + \frac{1}{d-g+1}\sum_{k=g}^{d}\Big(\mathrm{E}\Big[X_g^{k-1}\Big[T^{g,d,T^{-1}[k]}\Big]\Big] + \mathrm{E}\Big[X_{k-1}^{d}\Big[T^{g,d,T^{-1}[k]}\Big]\Big]\Big) \\
&= (d-g-1) + \frac{1}{d-g-1}\sum_{k=g}^{}\Big(\mathrm{E}\Big[X_0^{k-1-g}[\mathrm{id}]\Big] + \mathrm{E}\Big[X_0^{d-k-1}[\mathrm{id}]\Big]\Big).
\end{aligned}$$

Ceci est vrai pour tout $T \in \mathfrak{S}_n$ telle que $T(\llbracket g,d \rrbracket) \subseteq \llbracket g,d \rrbracket$, donc

$$\begin{aligned}
\mathrm{E}\big[X_g^d[\mathrm{id}]\big] &= (d-g-1) + \frac{1}{d-g-1}\sum_{k=g}\Big(\mathrm{E}\Big[X_0^{k-1-g}[\mathrm{id}]\Big] + \mathrm{E}\Big[X_0^{d-k-1}[\mathrm{id}]\Big]\Big) \\
&= \mathrm{E}\big[X_g^d[T]\big]
\end{aligned}$$

□

Annexe A. Hors-programme