

NOTES DE COURS

24 Décembre 2023

Table des matières

1. Algorithmes de Monte-Carlo	1
2. Arbres de jeux et principe de Yao	2
3. Inégalités de concentrations	6

1. Algorithmes de Monte-Carlo

Définition 1.1: Un algorithme de Monte-Carlo est un algorithme randomisé dont le **temps d'exécution** est **déterministe** mais le résultat peut être incorrect avec une certaine probabilité p (généralement petite)

- Pas de surprise sur la durée du calcul
- Peut-être faux mais c'est très rare
- Nous donne des algos rapides avec une faible probabilité d'échec

1.1. Biais

Un algorithme randomisé (eg: Monte-Carlo, Las-Vegas) peut être erroné - il répond OUI alors que la réponse aurait dû être NON - ce qui n'est pas le cas des algorithmes déterministes. Un algorithme de Monte-Carlo peut toujours renvoyer une réponse exacte si il a un **biais**.

Un algorithme biaisé vers le faux est toujours correct lorsqu'il retourne FAUX.

Un algorithme biaisé vers le vrai est toujours correct lorsqu'il retourne VRAI.

Définition 1.1.1: Un algorithme est *one-side error* lorsqu'il est soit biaisé vers le faux, soit biaisé vers le vrai

Définition 1.1.2: Un algorithme est *two-side error* lorsqu'il est biaisé vers le vrai et le faux

1.2. Classes de Complexités

À l'instar des algorithmes déterministes et des classes de complexités P et NP , les algorithmes randomisés ont également leurs classes de complexités.

1.2.1. BPP

Bounded-Error Probabilistic polynomial time → Le problème est résolvable en $\text{TIME}(P)$ avec un algorithme de Monte-Carlo non biaisé.

Le théorème d'Adelman énonce que BPP est inclus dans $P/poly$

1.3. RP

Randomized Polynomial Time → résolvable en $\text{TIME}(P)$ avec une probabilité bornée par un algorithme de Monte-Carlo biaisé.

1.4. ZPP

Zero-Error Probabilistic Polynomial Time → résolvable en $\text{TIME}(P)$ avec un algorithme de Las-Vegas.

$ZPP = RP \cap coRP$

1.5. PP

Polynomial Probabilistic → décidable en temps polynomial avec une probabilité d'erreur inférieure à 1/2.

1.6. Algorithme de Las-Vegas

Définition 1.6.1: Un algorithme de Las-Vegas est un algorithme randomisé dont l'exécution est **déterministe** mais le temps d'exécution est non-prévisible. Il s'arrête lorsqu'il trouve une réponse exacte

2. Arbres de jeux et principe de Yao

Vision "théorie des jeux" des algorithmes randomisés. Cela permet de visualiser des algorithmes probabilistes comme une distribution de probabilités sur des algorithmes déterministes.

Le **principe de Yao** permet d'établir une borne inférieure sur les performances d'un algorithme randomisé.

2.1. Arbre min-max

Forme la plus basique des algorithmes d'intelligence artificielle permettant de représenter un jeu entre deux joueursⁱ. Arbre où chaque valeur est binaire (l'input étant stocké dans les feuilles). Chaque noeud de l'arbre est soit un *min* (\wedge booléen) soit un *max* (\vee booléen).

Si le jeu n'était pas binaire, le concept serait le même, chaque feuilles contient une valeur numéraire. Les noeuds à distance paires du sommet seraient nommés MAX et les noeuds à une distance impaires MIN. L'évaluation de l'arbre suivrait donc le processus suivant: chaque noeud MAX retourne la valeur maximum parmi ses enfants et chaque noeud MIN la valeur minimum. Le but final étant - pour une certaine entrée - de donner la valeur de la racine.

On note T_k l'arbre de jeu de hauteur $2k$. La racine est un noeud \wedge , il y a donc exactement k noeuds \wedge et k noeuds \vee . L'arbre T_k possède donc $2^2 k = 4^k$ feuilles.

Évaluer un tel arbre nécessiterait donc, pour un algorithme déterministe, d'évaluer les 4^k noeuds. Cela dit, avec un algorithme probabiliste on peut augmenter la borne inférieure. Un exemple de tel algorithme est le suivant:

On commence par la racine et on évalue récursivement un des deux enfants du noeud courant. Si le noeud courant est un \wedge et que le résultat de l'analyse d'un de ces enfants est 0, il n'est pas nécessaire d'évaluer l'autre enfant. Selon le même principe, si le noeud courant est un \vee et qu'un des appel récursif retourne un 1 il n'est pas nécessaire d'évaluer l'autre enfant. Dans tous les autres cas, il est nécessaire de vérifier les deux enfants pour s'assurer que tout boume.

Théorème 2.1.1: Soit n'importe quelle instance d'un arbre de jeu binaire $T_{2,k}$, le nombre attendu d'étapes pour la stratégie d'évaluation randomisée est au plus 3^k

Preuve: On a 4 cas différents:

- Le noeud courant est un \vee :
 - Retourne 0: On a donc besoin de 2 appels récursifs, ce qui coûte

$$2 \times 3^{k-1}$$

lectures attendus.

- Retourne 1: On a, avec une probabilité d' $\frac{1}{2}$, seulement un enfant à évaluer.

$$\underbrace{\frac{1}{2} \times 2 \times 3^{k-1}}_{\text{Retourne 0}} + \underbrace{\frac{1}{2} \times 3^{k-1}}_{\text{Retourne 1}} = \frac{3}{2} \times 3^{k-1}$$

ⁱOn associe à chaque noeud un ensemble d'enfants représentant les options possible pour un des deux joueur

- Le noeud courant est un \wedge :
 - Retourne 0: Alors ses deux enfants retournent soit 0, avec une probabilité

$$2 \times 3^{k-1}$$

; soit qu'un seul des deux retourne 0, avec une probabilité

$$\frac{1}{2} \times 2 \times 3^{k-1} + \frac{1}{2} \times \left(\frac{3}{2} \times 3^{k-1} + 2 \times 3^{k-1} \right) = \frac{11}{4} \times 3^{k-1}$$

- Retourne 1: Alors ses deux enfants doivent retourner 1, il est nécessaire de regarder les deux enfants.

$$2 \times \underbrace{\frac{3}{2} \times 3^{k-1}}_{\text{Coût de vérification d'un noeud } \vee} = 3^k$$

Dans tous les cas, on remarque que le nombre attendu d'opération pour évaluer le \wedge racine ne dépasse pas 3^k

2.2. Théorie des jeux

Un jeu à somme nulle pour deux joueurs est un jeu pouvant être représenté par M , une matrice de gain $n \times m$ dont les entrées sont encodées par paires de stratégies, une pour chaque joueur. M_{ij} est la quantité payée par le joueur C (colonne) au joueur L (ligne) lorsque L choisit la stratégie i et C la stratégie j .

2.3. Stratégies mixtes

Dans le cas des stratégies mixtes, la matrice M ne contient plus les gains mais une distribution de probabilité sur l'ensemble des stratégies. Le gain attendu pour un paire de stratégie p, q est :

$$E[\text{payoff}] = p^T M q = \sum_{i=1}^n \sum_{j=1}^m p_i M_{ij} q_j$$

2.4. Théorème de Minmax

Théorème 2.4.1:

$$\max_p \min_q p^T M q = \min_q \max_p p^T M q$$

Il s'agit d'un principe fondamental de théorie des jeux.

Si l'on dénote V_L la meilleure borne inférieure sur le gain attendu par le joueur ligne en choisissant une stratégie mixte p , et par V_C la meilleure borne supérieure sur le gain payé par le joueur colonne pour une stratégie mixte q .

$$V_R = \max_p \min_q p^T M q$$

$$V_C = \min_q \max_p p^T M q$$

Par le Théorème 2.4.1, nous savons que $V_R = V_C$.

2.5. Principe de Yao

Soit un problème computationnel Π , on considère une collection finie \mathcal{A} d'algorithmes et un ensemble \mathcal{I} d'entrées, tel que tous les algos dans \mathcal{A} exécutent correctement le problème Π sur toutes les entrées dans \mathcal{I} . On écrit $C(I, A)$ le temps d'exécution d'un algorithme $A \in \mathcal{A}$ sur l'entrée $I \in \mathcal{I}$.

Cela définit un jeu à somme nulle entre deux joueurs: un designer d'algorithme et une entrée adverseⁱⁱ. L'ensemble \mathcal{A} peut-être vu comme un ensemble de stratégies pour le designer d'algorithme, une stratégie mixte peut alors être vue comme un algorithme randomisé. Dans le cas de l'ensemble \mathcal{I} , une stratégie mixte peut être vue comme une distribution sur les entrées possibles.

En appliquant le Théorème 2.4.1 on remarque que le temps d'exécution attendu pour un algorithme déterministe optimal sur une entrée choisie arbitrairement sur une distribution p est une borne inférieure sur le temps d'exécution de n'importe quel algorithme randomisé exact (Las-Vegas) pour Π .

Théorème 2.5.1: Pour toutes les distributions p sur \mathcal{I} et q sur \mathcal{A} , nous avons

$$\min_{A \in \mathcal{A}} E[C(I_p, A)] \leq \max_{I \in \mathcal{I}} E[C(I, A_q)]$$

Donc d'une part, nous avons $C(I_p, A)$ le temps d'exécution attendu pour le meilleur algorithme déterministeⁱⁱⁱ sur une entrée choisie aléatoirement. De l'autre côté, nous avons $C(I, A_q)$ le temps d'exécution d'un algorithme randomisé sur la pire entrée possible^{iv}.

Donc, pour prouver une borne inférieure sur la complexité d'un algorithme randomisé, il suffit de choisir n'importe quelle distribution d'entrée p et de prouver une borne inférieure sur le temps d'exécution attendu des algorithmes déterministes pour cette distribution p .

La force de cette technique est dans la flexibilité du choix de p et dans la réduction à une borne inférieure des algorithmes déterministes.

Attention, cela donne une borne inférieure sur les performances des algorithmes de Las-Vegas uniquement ! (C'est-à-dire ceux retournant une réponse exacte à tous les coups)

2.6. Principe de Yao pour les évaluation d'arbre de jeux

Si l'on reprend notre arbre de jeu AND-OR et son problème d'évaluation. Un algorithme randomisé peut-être vu comme une distribution de probabilité sur des algorithmes déterministes (car la longueur des calculs aussi bien que de nombre de choix à chaque étape sont toutes deux finies).

Imaginons un arbre de jeu T_k dont les feuilles se trouvent à une distance $2k$ de la racine, et dont tous les noeuds internes sont des NOR: Un noeud retourne un 1 uniquement si les deux entrées (enfants) sont à 0, sinon il retourne 0. L'analyse de cet arbre de NORs de hauteur $2k$ est la suivante:

Notons $p = \frac{3-\sqrt{5}}{2}$, chaque feuille de l'arbre contient 1 avec une probabilité p . Un noeud NOR retourne donc 1 avec une probabilité :

$$\left(\frac{\sqrt{5}-1}{2}\right)^2 = \frac{5+1-2\sqrt{5}}{4} = \frac{6+2\sqrt{5}}{4} = \frac{3-\sqrt{5}}{2} = p$$

La valeur de chaque noeud NOR est donc 1 avec une probabilité p et est indépendante des valeurs de tous les autres noeuds sur le même niveau.

Considérons maintenant un algorithme déterministe pour évaluer un tel arbre de jeu. Pour commencer, notons v un noeud de l'arbre dont l'algorithme cherche à obtenir la valeur. Il est simple de voir que l'on va d'abord évaluer un premier enfant de v avant d'aller voir l'autre enfant. On peut donc voir cela comme une *depth-first search* (recherche en profondeur) qui s'arrêterait d'évaluer les sous-arbre de v lorsque v sera évalué. Nous appelons un tel algorithme **depth-first pruning** (élagage en profondeur), en effet, les sous-arbres ne

ⁱⁱVoir ça comme de vrais adversaires, non pas qu'ils soient méchants entre eux, mais plus qu'ils veulent remporter ! Il n'est pas question de laisser à l'autre adversaire remporter la partie.

La théorie des jeux, c'est comme un Monopoly avec la famille, si il faut que ça se finisse dans les larmes et le sang ça se fera.

ⁱⁱⁱÉtant donné que l'algorithme est choisi par l'adversaire, il va clairement pas nous faire de cadeau en nous en donnant un pas top. Non, il prend le meilleur.

^{iv}Car choisie par l'adversaire, on oublie pas qu'il est pas là pour nous aider.

fournissant pas plus d'informations sur l'évaluation (par exemple, si le premier enfant retourne un 1 on sait qu'il est inutile d'aller évaluer l'autre enfant), on élague donc ce sous-arbre de l'évaluation.

Proposition 2.6.1: Si l'on prend maintenant un arbre NOR T dont les feuilles sont indépendamment définies à 1 avec une probabilité q pour une valeur fixée $q \in [0, 1]$. Notons $W(T)$ le nombre d'étapes minimum (parmi tous les algorithmes déterministes) nécessaires pour évaluer T . Il y a donc un algorithme de *depth-first pruning* dont le nombre d'étapes attendues pour évaluer T est $W(T)$

Cette Proposition 2.6.1 nous dit que, pour les besoins de notre borne inférieure, nous pouvons réduire notre attention aux algorithmes *depth-first pruning*. Nous définissons alors $W(h)$ comme étant le nombre attendu de feuilles inspectées par un tel algorithme d'élagage lorsqu'il évalue un noeud se trouvant à une distance h des feuilles, quand les feuilles sont définies indépendamment avec une probabilité $\frac{3-\sqrt{5}}{2}$. Alors le nombre de feuilles lues lors de l'exécution d'un tel algorithme est :

$$W(h) = \underbrace{W(h-1)}_{\text{Nombre d'étapes nécessaires pour évaluer le premier enfant}} + \underbrace{(1-p) \times W(h-1)}_{\substack{\text{Nombre d'étapes nécessaires} \\ \text{pour évaluer le second enfant} \\ \text{Cela n'a lieu que si le premier enfant vaut 0} \\ \text{ce qui arrive avec une probabilité } 1-p}}$$

étant donné que l'on doit effectuer un deuxième appel récursif uniquement si l'évaluation du premier enfant retourne 0, ce qui arrive avec une probabilité $(1-p)$. On peut alors résoudre la récurrence:

$$\begin{aligned} W(h) &= W(h-1) + (1-p) \times W(h-1) \\ &= (2-p) \times W(h-1) \\ &= (2-p)^h \\ &= (2-p)^{\log_2 n} \\ &= n^{\log_2(2-p)} \\ &\simeq n^{0.694} \end{aligned}$$

Sur base du principe de Yao, nous pouvons donc déduire le théorème suivant:

Théorème 2.6.1: Le temps d'exécution attendu pour n'importe quel algorithme randomisé évaluant toujours une instance de T_k correctement est au moins $n^{0.694}$, où $n = 2^2 k$ est le nombre de feuilles de T_k

2.7. Exercices d'examens

2.7.1. Exercice 2.2

Remarque préliminaire: Il s'agit d'une tentative de réponse il se peut qu'elle soit erronée !

→ On a un arbre d'hauteur h

→ Chaque noeud à 3 enfants

→ Il y a $n = 3^h$ feuilles contenant des valeurs booléennes

→ Chaque noeud retourne la valeur majoritaire parmi ses enfants (valeur booléenne donc)

1. Étant donné que l'on a 3 enfants et que les valeurs sont booléennes (0 ou 1), on sait qu'il ne peut y avoir d'égalité parmi la majorité (car nombre impair d'enfants). L'algorithme d'évaluation peut donc être défini comme suit: Pour évaluer un noeud v , on prend 2 de ses enfants aléatoirement et on les compare, si ils sont identiques c'est super, on a trouvé la majorité, v est donc évalué à la valeur des deux enfants sélectionnés, il n'est plus nécessaire d'évaluer le troisième étant donné qu'il n'impactera pas la majorité (qu'il soit identique ou différent). Si la valeur des deux enfants est différente, on va alors évaluer le troisième

pour trancher sur la majorité (et les trois enfants sont donc évalués).

Il est facile de remarquer qu'il n'y a que deux cas possibles:

- Les enfants sont 3 b^v : la majorité est obtenue lorsqu'on évalue 2 enfants. Le nombre d'évaluation est donc de 2.
- Les enfants sont 2 b et 1 $\neg b$: Soit on sélectionne les deux b du premier coup et c'est niquel, soit on sélectionne un b et le $\neg b$ et dans ce cas là il faut évaluer le dernier enfant pour savoir quelle est la valeur majoritaire. Le nombre d'évaluation est donc de

$$\underbrace{\frac{1}{3} \times 2}_{2b \text{ du premier coup}} + \underbrace{\frac{1}{3} \times (2 + 1)}_{1b \text{ et } 1 \neg b \text{ puis un } b} = \frac{5}{3}$$

Le pire cas est donc le premier cas = 2 donc $T(h) = 2T(h-1) = 2^h$, $n = 3^h \Rightarrow h = \log_3 n$, $2^{\log_3 n} = n^{\log_3 2}$

- On part du principe que chaque feuilles prend la valeur 1 avec une probabilité $p = \frac{1}{2}$. Notons $W(h)$ le nombre de feuilles lues lorsqu'on analyse un noeud se trouvant à une distance h des feuilles. On sait qu'il est nécessaire d'évaluer à minima 2 enfants ($2W(h-1)$) et qu'il y a une probabilité $\frac{1}{2}$ que les deux valeurs soient différentes car:

- Proba d'avoir 2 valeurs identiques \otimes : $\underbrace{\frac{1}{2} \times \frac{1}{2}}_{\text{On a deux } 1} + \underbrace{\frac{1}{2} \times \frac{1}{2}}_{\text{On a deux } 0} = \frac{1}{4} + \frac{1}{4} = \frac{1}{2}$
- Proba d'avoir 2 valeurs différentes : $\underbrace{1 - \frac{1}{2}}_{\otimes} = \frac{1}{2}$

On sait alors qu'il est nécessaire d'effectuer $\frac{1}{2}W(h-1)$ évaluations.

$$\begin{aligned} W(h) &= 2W(h-1) + \frac{1}{2}W(h-1) \\ &= \frac{5}{2}W(h-1) \\ &= \left(\frac{5}{2}\right)^h = \left(\frac{5}{2}\right)^{\log_3 n} \\ &= n^{\log_3 5} \simeq n^{1.464} \end{aligned}$$

3. Inégalités de concentrations

Définition 3.1: En probabilité, les **inégalités de concentration** fournissent des bornes sur la probabilité qu'une variable aléatoire dévie d'une certaine valeur (généralement l'espérance de cette variable aléatoire)

En d'autres termes, ces inégalités nous donnent des bornes sur les variations que peuvent prendre une variable aléatoire. Dans le cas des algorithmes randomisés celles-ci sont intéressantes pour prouver qu'un tel algorithme nous donne une réponse correcte avec une grande probabilité.

3.1. Inégalité de Markov

Il s'agit de l'inégalité la plus simple. On l'utilisera comme base pour d'autres inégalités. Elle utilise l'espérance de la variable aléatoire.

Définition 3.1.1: Si X est une variable aléatoire et $f(x)$ une fonction réelle. Alors l'**espérance** de $f(x)$ est donnée par

$$E[f(X)] = \sum_{x \in X} f(x)P[X = x]$$

^v b est une valeur booléenne, on se moque de laquelle.

L'inégalité de Markov nous donne la borne la plus serrée possible quand nous savons que X est non-négatif et a une espérance donnée.

Théorème 3.1.1: Si X est une variable aléatoire non-négative et $a > 0$ alors

$$P[X \geq a] \leq \frac{E[X]}{a}$$

Et de manière équivalente

$$P[X \geq kE[X]] \leq \frac{1}{k}$$

Preuve: Définissons une fonction

$$f(x) = \begin{cases} 1 & \text{si } x \geq a \\ 0 & \text{sinon} \end{cases}$$

Alors $P[X \geq a] = E[f(X)]$. Étant donné que $f(x) \leq \frac{x}{a}$ pour n'importe quel x , on a

$$E[f(X)] \leq E\left[\frac{X}{a}\right] = \frac{E[X]}{a}$$

Malheureusement, cette borne est trop faible pour apporter des résultats utiles. Cependant, elle peut être utilisée pour trouver de meilleures bornes sur la probabilité de la queue en utilisant plus d'informations sur la distribution de la variable aléatoire.

3.2. Inégalité de Chebyshev

Cette inégalité est basée sur la connaissance de la variance de la distribution.

Définition 3.2.1: Pour une variable aléatoire X avec une espérance μ_X , sa **variance** est définie comme

$$\sigma_X^2 = E[(X - \mu_X)^2]$$

Définition 3.2.2: L'**écart-type** d'une variable aléatoire X , notée σ_X est la racine carrée positive de la variance de X .

Théorème 3.2.1: Soit X une variable aléatoire ayant une espérance μ_X et un écart-type σ_X , pour n'importe quel $t \in \mathbb{R}^+$,

$$P[|X - \mu_X| \geq t\sigma_X] \leq \frac{1}{t^2}$$

Preuve: On observe que

$$P[|X - \mu_X| \geq t\sigma_X] = P[(X - \mu_X)^2 \geq t^2\sigma_X^2]$$

Définissons une variable aléatoire $Y = (X - \mu_X)^2$, son espérance vaut alors $E[Y] = \sigma_X^2$, si l'on applique l'inégalité de Markov à cette variable aléatoire avec $a = t^2$, clairement^{vi}:

^{vi}Je dis clairement, mais j'avais tellement pas compris la preuve (qui pourtant est super simple) que j'ai passé 1h à la refaire. Oui parfois y'a des trucs débiles sur lesquels on passe une plombe, pour la peine je vais écrire les étapes.

$$\begin{aligned}
P[Y \geq t^2] &\leq \frac{E[Y]}{t^2} \\
P[(X - \mu)^2 \geq t^2] &\leq \frac{\sigma^2}{t^2} \\
P[(X - \mu)^2 \geq \sigma^2 t^2] &\leq \frac{1}{t^2} \\
P[(X - \mu) \geq \sigma t] &\leq \frac{1}{t^2}
\end{aligned}$$

3.3. Borne de Chernoff

Borne sur la queue de la distribution bien plus précises que celles de Markov ou Chebyshev. Ces bornes sont utilisées dans le cas où les variables aléatoires ne peuvent pas être modélisées comme une somme de variables aléatoires indépendantes (par exemple des variables de Bernoulli^{vii}).

Définition 3.3.1: Soit X_1, \dots, X_n des **variables aléatoires indépendantes de Bernoulli** tel que $1 \leq i \leq n$, $P[X_i = 1] = p$ et $P[X_i = 0] = (1 - p)$. Si l'on définit $X = \sum_{i=1}^n X_i$, alors X suit une distribution binomiale.

Définition 3.3.2: Des essais de **Poisson** font références à des variables aléatoires indépendantes dont les probabilités sont propres aux variables. Il s'agit d'une généralisation de Bernoulli. Soit X_1, \dots, X_n des variables de Bernoulli tel que pour $1 \leq i \leq n$, $P[X_i = 1] = p_i$ et $P[X_i = 0] = (1 - p_i)$

Nous nous concentrons sur le cas général (Poisson) mais il fonctionne bien évidemment aussi dans le cas "Bernoulli" où toutes les variables partagent la même probabilité p . On peut se poser des questions sur la déviation de l'espérance μ de X , $\mu = \sum_i = 1^n p_i$ telles que "Quelle est la probabilité que X dépasse $(1 + \delta)\mu$?". Une réponse à cette question est utile pour analyser un algorithme randomisé, cela montre que les chances que l'algorithme échoue une certaine performances sont faibles. Un autre type de question que l'on peut se poser est "Quelle valeur maximale δ peut prendre de sorte que la probabilité de la queue soit plus petite qu'une valeur ε ?"

Théorème 3.3.1: Soit X_1, \dots, X_n une collection d'essais de Poissons indépendantes^{viii} tq pour tout $1 \leq i \leq n$, $P[X_i = 1] = p_i$ où $0 < p_i < 1$. Alors, pour un $X = \sum_{i=1}^n X_i$, $\mu = E[X] = \sum_i p_i$ et un $\delta > 0$,

$$P[X > (1 + \delta)\mu] < \left[\frac{e^\delta}{(1 + \delta)^{1+\delta}} \right]^\mu$$

Remarque: le membre de droite de l'inégalité est une fonction ne dépendant que des paramètres δ et μ

L'idée derrière cette formule est d'appliquer les inégalités de Markov sur les *fonction génératrices des moments* des variables aléatoires X définies comme $M_{X(t)} = E[e^{tX}]$.

Preuve:

^{vii}Résultat d'un pile ou face

^{viii}Il s'agit donc simplement de variables aléatoires indépendantes de Bernoulli x_i ayant chacune une probabilité p_i

$$\begin{aligned}
P[X > (1 + \delta)\mu] &= P[e^{tX} > e^{t(1+\delta)\mu}] \\
\Rightarrow P[e^{tX} > e^{t(1+\delta)\mu}] &< \frac{E[e^{tX}]}{e^{t(1+\delta)\mu}} \quad \text{Inégalité de Markov}
\end{aligned}$$

Ici, une petite étape transformation du numérateur dans le terme de droite s'impose

$$\begin{aligned}
E[e^{tX}] &= E[e^{t \sum_i x_i}] \\
&= E\left[\prod_i e^{tx_i}\right] \quad \text{Propriété des exposants} \\
&= \prod_i E[e^{tx_i}] \quad \text{Linéarité de l'espérance}
\end{aligned}$$

Ici aussi, une petite transformation est nécessaire pour nous simplifier la vie, elle est pas super triviale pour un étudiant en info qui comprend pas grand chose en proba. En gros, l'idée est quand même simple mais il faut l'avoir: l'espérance représentant la valeur que peut prendre une variable aléatoire et qu'on a des variables de Poissons (c'est à dire dont les valeurs x_i ont chacune une probabilité p_i), on peut observer que e^{tX_i} prend la valeur e^t avec une proba p_i et 1 avec une proba $1 - p_i$, en effet:

- si la proba est p_i alors $X = 1$ et donc $e^{t1} = e^t$ et,
- si la proba est $1 - p_i$ alors $X_i = 0$ et $e^{t0} = 1$.

On peut donc effectuer la transformation suivante:

$$\begin{aligned}
E[e^{tX_i}] &= e^t p_i + (1 - p_i) \\
&= 1 + p_i(e^t - 1)
\end{aligned}$$

Enfin, on va encore faire une ultime transformation, bien moins triviale; elle se base sur l'inégalité bien connue^{ix} : $1 + x < e^x$

$$\begin{aligned}
\prod_{i(1+p_i(e^t-1))} &< \prod_{i(e^{p_i(e^t-1)})} \\
&= e^{\sum_i p_i(e^t-1)} \quad \text{Propriétés des exposants} \\
&= e^{(e^t-1)\mu} \quad \text{car } \sum_i p_i = \mu
\end{aligned}$$

Remettons toutes ces transformations dans l'équation initiale:

$$\begin{aligned}
P[e^{tX} > e^{t(1+\delta)\mu}] &< \prod_i \frac{1 + p_i(e^t - 1)}{e^{t(1+\delta)\mu}} \\
&< \frac{e^{(e^t-1)\mu}}{e^{t(1+\delta)\mu}}
\end{aligned}$$

On doit donc maintenant une valeur pour t de sorte à obtenir la meilleure borne possible, pour ça on a juste à prendre, il s'agit d'un problème d'optimisation simple sans contraintes, on prend la valeur obtenue lorsque la dérivée est nulle. Cela donne

$$\frac{d}{dt} \left(\frac{e^{(e^t-1)\mu}}{e^{t(1+\delta)\mu}} \right) = \mu * (e^t - \delta - 1) * e^{\mu * (e^t-1) - (\delta+1) * \mu * t} = 0$$

La valeur optimale est $e^t = \delta + 1 \Rightarrow t = \ln(\delta + 1)$ pour $\delta > 0$, on a donc notre borne.

^{ix}Non