

Cours 7: Classes Probabilistes

Olivier Bournez

bournez@lix.polytechnique.fr
LIX, Ecole Polytechnique

Au menu

Retours sur quelques algorithmes

Classes probabilistes

Quelques résultats

Sous menu

Retours sur quelques algorithmes

Calcul de la médiane

Bornes de Chernoff

Application : Test d'hypothèses

Application : Calcul de la médiane

- Étant donné un ensemble $S = \{s_1, s_2, \dots, s_n\}$ de n éléments parmi un univers totalement ordonné,
 - ▶ on appelle *médiane* de S un élément m de S tel que
 1. au moins $\lfloor n/2 \rfloor$ éléments de S soit inférieurs ou égaux à m ,
 2. et au moins $\lfloor n/2 \rfloor + 1$ éléments de S soit supérieurs à m .

Application : Calcul de la médiane

- Étant donné un ensemble $S = \{s_1, s_2, \dots, s_n\}$ de n éléments parmi un univers totalement ordonné,
 - ▶ on appelle *médiane* de S un élément m de S tel que
 1. au moins $\lfloor n/2 \rfloor$ éléments de S soit inférieurs ou égaux à m ,
 2. et au moins $\lfloor n/2 \rfloor + 1$ éléments de S soit supérieurs à m .
- Autrement dit, si S est trié, m est l'élément d'indice $\lceil n/2 \rceil$.

Application : Calcul de la médiane

- Étant donné un ensemble $S = \{s_1, s_2, \dots, s_n\}$ de n éléments parmi un univers totalement ordonné,
 - ▶ on appelle *médiane* de S un élément m de S tel que
 1. au moins $\lfloor n/2 \rfloor$ éléments de S soit inférieurs ou égaux à m ,
 2. et au moins $\lfloor n/2 \rfloor + 1$ éléments de S soit supérieurs à m .
- Autrement dit, si S est trié, m est l'élément d'indice $\lceil n/2 \rceil$.
- Bien entendu, l'idée est d'utiliser un algorithme plus efficace que $\mathcal{O}(n \log(n))$, ce qui peut être atteint en triant les données, puis en retournant cet élément.

Application : Calcul de la médiane

- Étant donné un ensemble $S = \{s_1, s_2, \dots, s_n\}$ de n éléments parmi un univers totalement ordonné,
 - ▶ on appelle *médiane* de S un élément m de S tel que
 1. au moins $\lfloor n/2 \rfloor$ éléments de S soit inférieurs ou égaux à m ,
 2. et au moins $\lfloor n/2 \rfloor + 1$ éléments de S soit supérieurs à m .
- Autrement dit, si S est trié, m est l'élément d'indice $\lfloor n/2 \rfloor$.
- Bien entendu, l'idée est d'utiliser un algorithme plus efficace que $\mathcal{O}(n \log(n))$, ce qui peut être atteint en triant les données, puis en retournant cet élément.
- On connaît une solution déterministe en $\mathcal{O}(n)$ opérations. On présente ici une version randomisée simple de même complexité.

- L'idée de l'algorithme est d'utiliser de l'échantillonnage :

- L'idée de l'algorithme est d'utiliser de l'échantillonnage :
 - ▶ on essaye de trouver deux éléments d et u tels que $d \leq m \leq u$,

- L'idée de l'algorithme est d'utiliser de l'échantillonnage :
 - ▶ on essaye de trouver deux éléments d et u tels que $d \leq m \leq u$,
 - ▶ et tel que le nombre d'éléments entre d et u soit faible, c'est-à-dire en $o(n/\log(n))$.

- L'idée de l'algorithme est d'utiliser de l'échantillonnage :
 - ▶ on essaye de trouver deux éléments d et u tels que $d \leq m \leq u$,
 - ▶ et tel que le nombre d'éléments entre d et u soit faible, c'est-à-dire en $o(n/\log(n))$.

- Notons $C = \{s \mid d \leq s \leq u\}$ les éléments entre d et u .

- L'idée de l'algorithme est d'utiliser de l'échantillonnage :
 - ▶ on essaye de trouver deux éléments d et u tels que $d \leq m \leq u$,
 - ▶ et tel que le nombre d'éléments entre d et u soit faible, c'est-à-dire en $o(n/\log(n))$.
- Notons $C = \{s \mid d \leq s \leq u\}$ les éléments entre d et u .
- Si l'on arrive à trouver deux éléments comme cela, il est facile de trouver la médiane en temps linéaire :

- L'idée de l'algorithme est d'utiliser de l'échantillonnage :
 - ▶ on essaye de trouver deux éléments d et u tels que $d \leq m \leq u$,
 - ▶ et tel que le nombre d'éléments entre d et u soit faible, c'est-à-dire en $o(n/\log(n))$.

- Notons $C = \{s \mid d \leq s \leq u\}$ les éléments entre d et u .

- Si l'on arrive à trouver deux éléments comme cela, il est facile de trouver la médiane en temps linéaire :
 - ▶ on parcourt la liste S , et

- L'idée de l'algorithme est d'utiliser de l'échantillonnage :
 - ▶ on essaye de trouver deux éléments d et u tels que $d \leq m \leq u$,
 - ▶ et tel que le nombre d'éléments entre d et u soit faible, c'est-à-dire en $o(n/\log(n))$.
- Notons $C = \{s \mid d \leq s \leq u\}$ les éléments entre d et u .
- Si l'on arrive à trouver deux éléments comme cela, il est facile de trouver la médiane en temps linéaire :
 - ▶ on parcourt la liste S , et
 - ▶ on compte le nombre ℓ_d d'éléments inférieurs à d ,

- L'idée de l'algorithme est d'utiliser de l'échantillonnage :
 - ▶ on essaye de trouver deux éléments d et u tels que $d \leq m \leq u$,
 - ▶ et tel que le nombre d'éléments entre d et u soit faible, c'est-à-dire en $o(n/\log(n))$.
- Notons $C = \{s \mid d \leq s \leq u\}$ les éléments entre d et u .
- Si l'on arrive à trouver deux éléments comme cela, il est facile de trouver la médiane en temps linéaire :
 - ▶ on parcourt la liste S , et
 - ▶ on compte le nombre ℓ_d d'éléments inférieurs à d ,
 - ▶ et on trie l'ensemble C : puisque $|C| = o(n/\log(n))$, trier C se fait en temps $o(n)$ par n'importe quel algorithme de tri qui fonctionne en temps $\mathcal{O}(m \log(m))$ pour m éléments.

- L'idée de l'algorithme est d'utiliser de l'échantillonnage :
 - ▶ on essaye de trouver deux éléments d et u tels que $d \leq m \leq u$,
 - ▶ et tel que le nombre d'éléments entre d et u soit faible, c'est-à-dire en $o(n/\log(n))$.
- Notons $C = \{s \mid d \leq s \leq u\}$ les éléments entre d et u .
- Si l'on arrive à trouver deux éléments comme cela, il est facile de trouver la médiane en temps linéaire :
 - ▶ on parcourt la liste S , et
 - ▶ on compte le nombre ℓ_d d'éléments inférieurs à d ,
 - ▶ et on trie l'ensemble C : puisque $|C| = o(n/\log(n))$, trier C se fait en temps $o(n)$ par n'importe quel algorithme de tri qui fonctionne en temps $\mathcal{O}(m \log(m))$ pour m éléments.
 - ▶ L'élément d'indice $\lfloor n/2 \rfloor - \ell_d + 1$ de C est alors m .

- Pour trouver d et u , on échantillonne avec remplacement un (multi-)ensemble R de $\lceil n^{3/4} \rceil$ éléments :

- Pour trouver d et u , on échantillonne avec remplacement un (multi-)ensemble R de $\lceil n^{3/4} \rceil$ éléments :
 - ▶ c'est-à-dire que l'on pioche au hasard uniformément ce nombre d'éléments parmi S .

- Pour trouver d et u , on échantillonne avec remplacement un (multi-)ensemble R de $\lceil n^{3/4} \rceil$ éléments :
 - ▶ c'est-à-dire que l'on pioche au hasard uniformément ce nombre d'éléments parmi S .
- On souhaite que chaque étape fonctionne avec *grande probabilité*,

- Pour trouver d et u , on échantillonne avec remplacement un (multi-)ensemble R de $\lceil n^{3/4} \rceil$ éléments :
 - ▶ c'est-à-dire que l'on pioche au hasard uniformément ce nombre d'éléments parmi S .
- On souhaite que chaque étape fonctionne avec *grande probabilité*,
 - ▶ c'est-à-dire avec une probabilité au moins $1 - \mathcal{O}(1/n^c)$ pour une constante c .

- Pour trouver d et u , on échantillonne avec remplacement un (multi-)ensemble R de $\lceil n^{3/4} \rceil$ éléments :
 - ▶ c'est-à-dire que l'on pioche au hasard uniformément ce nombre d'éléments parmi S .
- On souhaite que chaque étape fonctionne avec *grande probabilité*,
 - ▶ c'est-à-dire avec une probabilité au moins $1 - \mathcal{O}(1/n^c)$ pour une constante c .
- Pour garantir qu'avec grande probabilité m soit entre d et u , on fixe d et u comme étant respectivement les $\lfloor n^{3/4}/2 - \sqrt{n} \rfloor$ ème et $\lfloor n^{3/4}/2 + \sqrt{n} \rfloor$ ème éléments de R .

L'algorithme

1. Choisir $\lceil n^{3/4} \rceil$ éléments dans S , avec un tirage uniforme et avec remise. Soit R les éléments obtenus.
2. Trier R .
3. Soit d le $\lfloor n^{3/4}/2 - \sqrt{n} \rfloor$ élément de R .
4. Soit u le $\lfloor n^{3/4}/2 + \sqrt{n} \rfloor$ élément de R .
5. En comparant chaque élément de S à d et u , calculer les ensembles $C = \{x \in S \mid d \leq x \leq u\}$, $\ell_d = |\{x \in S \mid x < d\}|$ et $\ell_u = |\{x \in S \mid x > u\}|$.
6. Si $\ell_d > n/2$ ou $\ell_u > n/2$ alors échouer.
7. Si $|C| \leq 4n^{3/4}$ alors trier S sinon échouer.
8. Retourner le $\lfloor n/2 \rfloor - \ell_d + 1$ élément de C .

Proposition

La probabilité que l'algorithme échoue est en $\mathcal{O}(n^{-1/4})$.

- On considère les trois événements

$$E_1 : Y_1 = |\{r \in R | r \leq m\}| < n^{3/4}/2 - \sqrt{n}$$

$$E_2 : Y_2 = |\{r \in R | r \geq m\}| < n^{3/4}/2 - \sqrt{n}$$

$$E_3 : |C| > 4n^{3/4}.$$

- On considère les trois événements

$$E_1 : Y_1 = |\{r \in R | r \leq m\}| < n^{3/4}/2 - \sqrt{n}$$

$$E_2 : Y_2 = |\{r \in R | r \geq m\}| < n^{3/4}/2 - \sqrt{n}$$

$$E_3 : |C| > 4n^{3/4}.$$

- L'algorithme termine si au moins l'un des trois événements E_1 , E_2 ou E_3 se produit :

- On considère les trois événements

$$E_1 : Y_1 = |\{r \in R | r \leq m\}| < n^{3/4}/2 - \sqrt{n}$$

$$E_2 : Y_2 = |\{r \in R | r \geq m\}| < n^{3/4}/2 - \sqrt{n}$$

$$E_3 : |C| > 4n^{3/4}.$$

- L'algorithme termine si au moins l'un des trois événements E_1 , E_2 ou E_3 se produit :
 - ▶ en effet, un échec à l'étape 7 correspond à l'événement E_3 .

- On considère les trois événements

$$E_1 : Y_1 = |\{r \in R | r \leq m\}| < n^{3/4}/2 - \sqrt{n}$$

$$E_2 : Y_2 = |\{r \in R | r \geq m\}| < n^{3/4}/2 - \sqrt{n}$$

$$E_3 : |C| > 4n^{3/4}.$$

- L'algorithme termine si au moins l'un des trois événements E_1 , E_2 ou E_3 se produit :
 - ▶ en effet, un échec à l'étape 7 correspond à l'événement E_3 .
 - ▶ Un échec à l'étape 6 implique $\ell_d > n/2$ ou $\ell_u > n/2$.

- On considère les trois événements

$$E_1 : Y_1 = |\{r \in R | r \leq m\}| < n^{3/4}/2 - \sqrt{n}$$

$$E_2 : Y_2 = |\{r \in R | r \geq m\}| < n^{3/4}/2 - \sqrt{n}$$

$$E_3 : |C| > 4n^{3/4}.$$

- L'algorithme termine si au moins l'un des trois événements E_1 , E_2 ou E_3 se produit :
 - ▶ en effet, un échec à l'étape 7 correspond à l'événement E_3 .
 - ▶ Un échec à l'étape 6 implique $\ell_d > n/2$ ou $\ell_u > n/2$.
 - ▶ Chacun de ces cas est équivalent à E_1 ou E_2 .

- On considère les trois événements

$$E_1 : Y_1 = |\{r \in R | r \leq m\}| < n^{3/4}/2 - \sqrt{n}$$

$$E_2 : Y_2 = |\{r \in R | r \geq m\}| < n^{3/4}/2 - \sqrt{n}$$

$$E_3 : |C| > 4n^{3/4}.$$

- L'algorithme termine si au moins l'un des trois événements E_1 , E_2 ou E_3 se produit :
 - ▶ en effet, un échec à l'étape 7 correspond à l'événement E_3 .
 - ▶ Un échec à l'étape 6 implique $\ell_d > n/2$ ou $\ell_u > n/2$.
 - ▶ Chacun de ces cas est équivalent à E_1 ou E_2 .
- Par une union-bound, il suffit de montrer que $\Pr(E_i)$ est en $\mathcal{O}(n^{-1/4})$.

- On a

$$\Pr(E_1) \leq \frac{1}{4} n^{-1/4}.$$

■ On a

$$\Pr(E_1) \leq \frac{1}{4} n^{-1/4}.$$

- ▶ En effet, considérons la variable aléatoire X_i qui vaut 1 si le i ème échantillon est inférieur ou égal à la médiane m , et 0 sinon.
- ▶ Les X_i sont indépendants, puisque l'on procède avec remplacement.

■ On a

$$\Pr(E_1) \leq \frac{1}{4} n^{-1/4}.$$

- ▶ En effet, considérons la variable aléatoire X_i qui vaut 1 si le i ème échantillon est inférieur ou égal à la médiane m , et 0 sinon.
- ▶ Les X_i sont indépendants, puisque l'on procède avec remplacement.
- ▶ Puisqu'il y a $(n-1)/2 + 1$ éléments inférieurs ou égaux à l , on a $\Pr(X_i = 1) = \frac{(n-1)/2+1}{n} = \frac{1}{2} + \frac{1}{2n}$.

■ On a

$$\Pr(E_1) \leq \frac{1}{4} n^{-1/4}.$$

- ▶ En effet, considérons la variable aléatoire X_i qui vaut 1 si le i ème échantillon est inférieur ou égal à la médiane m , et 0 sinon.
- ▶ Les X_i sont indépendants, puisque l'on procède avec remplacement.
- ▶ Puisqu'il y a $(n-1)/2 + 1$ éléments inférieurs ou égaux à l , on a $\Pr(X_i = 1) = \frac{(n-1)/2+1}{n} = \frac{1}{2} + \frac{1}{2n}$.
- ▶ L'événement E_1 est équivalent à

$$Y_1 = \sum_{i=1}^{n^{3/4}} X_i < \frac{1}{2} n^{3/4} - \sqrt{n}.$$

Y_1 est la somme de tirages de Bernoulli, il suit donc une loi binomiale de paramètres $n^{3/4}$ et $1/2 + 1/2n$.

- On a

$$\Pr(E_1) \leq \frac{1}{4} n^{-1/4}.$$

- En effet, considérons la variable aléatoire X_i qui vaut 1 si le i ème échantillon est inférieur ou égal à la médiane m , et 0 sinon.
- Les X_i sont indépendants, puisque l'on procède avec remplacement.
- Puisqu'il y a $(n-1)/2 + 1$ éléments inférieurs ou égaux à l , on a $\Pr(X_i = 1) = \frac{(n-1)/2+1}{n} = \frac{1}{2} + \frac{1}{2n}$.
- L'événement E_1 est équivalent à

$$Y_1 = \sum_{i=1}^{n^{3/4}} X_i < \frac{1}{2} n^{3/4} - \sqrt{n}.$$

Y_1 est la somme de tirages de Bernoulli, il suit donc une loi binomiale de paramètres $n^{3/4}$ et $1/2 + 1/2n$.

- L'inégalité de Tchebychev donne alors

$$\begin{aligned} \Pr(E_1) &= \Pr(Y_1 < \frac{1}{2} n^{3/4} - \sqrt{n}) \\ &\leq \Pr(|Y_1 - E[Y_1]| > \sqrt{n}) \\ &\leq \frac{\text{Var}[Y_1]}{n} < \frac{1/4 n^{3/4}}{n} \\ &= \frac{1}{4} n^{-1/4}. \end{aligned}$$

- On a

$$\Pr(E_3) \leq \frac{1}{2} n^{-1/4}.$$

■ On a

$$\Pr(E_3) \leq \frac{1}{2} n^{-1/4}.$$

- ▶ si E_3 se produit, soit au moins $2n^{3/4}$ éléments sont plus grands que m , soit au moins $2n^{3/4}$ sont plus petits que m .
- ▶ Bornons le premier, puisque l'autre est symétrique.
- ▶ L'ordre de u dans l'ensemble S trié doit être au moins $\frac{1}{2}n + 2n^{3/4}$, et donc R possède au moins $\frac{1}{2}n^{3/4} - \sqrt{n}$ éléments parmi les $\frac{1}{2}n - 2n^{3/4}$ éléments les plus grands de S .
- ▶ Soit X le nombre d'éléments parmi les $\frac{1}{2}n - 2n^{3/4}$ éléments les plus grands de S .
- ▶ X s'écrit $X = \sum_{i=1}^{n^{3/4}} X_i$, où X_i vaut 1 si le i ème élément pioché est parmi les $\frac{1}{2}n - 2n^{3/4}$ éléments les plus grands de S , et 0 sinon.
- ▶ X , somme de lois de Bernoulli, suit une loi binomiale, et l'inégalité de Tchebychev permet de majorer la probabilité de cet événement par

$$\begin{aligned} \Pr(X \geq \frac{1}{2}n^{3/4} - \sqrt{n}) &\leq \Pr(|X - E[X]| \geq \sqrt{n}) \\ &\leq \frac{\text{Var}[X]}{n} \\ &< \frac{\frac{n}{4n^{3/4}}}{n} \\ &= \frac{1}{4} n^{-1/4}. \end{aligned}$$

De Monte Carlo à Las Vegas

- Observons qu'en répétant cet algorithme jusqu'à ce qu'il réussisse, on obtient un algorithme de *Las Vegas* :

De Monte Carlo à Las Vegas

- Observons qu'en répétant cet algorithme jusqu'à ce qu'il réussisse, on obtient un algorithme de *Las Vegas* :
 - ▶ il retournerait toujours une réponse correcte, mais son temps de réponse est aléatoire.

De Monte Carlo à Las Vegas

- Observons qu'en répétant cet algorithme jusqu'à ce qu'il réussisse, on obtient un algorithme de *Las Vegas* :
 - ▶ il retournerait toujours une réponse correcte, mais son temps de réponse est aléatoire.
- En fait, le nombre d'itérations de l'algorithme serait alors donné par une loi géométrique.

De Monte Carlo à Las Vegas

- Observons qu'en répétant cet algorithme jusqu'à ce qu'il réussisse, on obtient un algorithme de *Las Vegas* :
 - ▶ il retournerait toujours une réponse correcte, mais son temps de réponse est aléatoire.
- En fait, le nombre d'itérations de l'algorithme serait alors donné par une loi géométrique.
- On peut assez facilement se convaincre que l'algorithme obtenu fonctionnerait en temps moyen linéaire.

Sous menu

Retours sur quelques algorithmes

Calcul de la médiane

Bornes de Chernoff

Application : Test d'hypothèses

Bornes de Chernoff : au dessus de la moyenne

Théorème

On considère des variables X_1, X_2, \dots, X_n à valeurs dans $\{0, 1\}$ indépendantes telles que $\Pr(X_i) = p_i$. Soit $X = \sum_{i=1}^n X_i$, et $\mu = E[X]$. Alors on a les bornes de Chernoff suivantes.

- Pour tout $\delta > 0$

$$\Pr(X \geq (1 + \delta)\mu) \leq \left(\frac{e^\delta}{(1 + \delta)^{1+\delta}} \right)^\mu.$$

- Pour tout $0 < \delta \leq 1$

$$\Pr(X \geq (1 + \delta)\mu) \leq e^{-\mu\delta^2/3}.$$

- Pour $R \geq 6\mu$,

$$\Pr(X \geq R) \leq 2^{-R}.$$

Bornes de Chernoff : au dessus de la moyenne

Théorème

On considère des variables X_1, X_2, \dots, X_n à valeurs dans $\{0, 1\}$ indépendantes telles que $\Pr(X_i) = p_i$. Soit $X = \sum_{i=1}^n X_i$, et $\mu = E[X]$. Alors on a les bornes de Chernoff suivantes.

- Pour tout $0 < \delta < 1$

$$\Pr(X \leq (1 - \delta)\mu) \leq \left(\frac{e^{-\delta}}{(1 - \delta)^{1-\delta}} \right)^\mu ;$$

- Pour tout $0 < \delta < 1$

$$\Pr(X \leq (1 - \delta)\mu) \leq e^{-\mu\delta^2/2}.$$

Sous menu

Retours sur quelques algorithmes

Calcul de la médiane

Bornes de Chernoff

Application : Test d'hypothèses

Application : Test d'hypothèses

- Supposons que l'on veuille évaluer une probabilité p de mutation de virus dans une population.
- Étant donné un virus, on peut déterminer s'il a muté, mais le test est coûteux.
- Supposons que l'on fasse le test sur une population de taille n , et que l'on observe que parmi cet échantillon que $X = \bar{p}n$ virus ont muté.
- Si n est suffisamment grand, on peut espérer que \bar{p} est proche de p .

- Formellement, cette intuition est capturée par la notion d'intervalle de confiance.

- **Definition (Intervalle de confiance)**

Un $1 - \gamma$ -*intervalle de confiance* pour un paramètre p est un intervalle $[\bar{p} - \delta, \bar{p} + \delta]$ tel que

$$\Pr(p \in [\bar{p} - \delta, \bar{p} + \delta]) \geq 1 - \gamma.$$

Théorème

On peut déterminer un $1 - \gamma$ -intervalle de confiance en utilisant un échantillon de taille

$$n = \lceil 3 \frac{1}{\delta^2} \ln \frac{2}{\gamma} \rceil.$$

Preuve

- On considère $X = \sum_i X_i$, avec $\Pr(X_i = 1) = p$, $\Pr(X_i = 0) = 1 - p$.

Preuve

- On considère $X = \sum_i X_i$, avec $\Pr(X_i = 1) = p$, $\Pr(X_i = 0) = 1 - p$.
- Sur n expériences, on aura $E[X] = np$, et donc on va considérer $\bar{p} = X/n$ comme estimation de p .

Preuve

- On considère $X = \sum_i X_i$, avec $\Pr(X_i = 1) = p$, $\Pr(X_i = 0) = 1 - p$.
- Sur n expériences, on aura $E[X] = np$, et donc on va considérer $\bar{p} = X/n$ comme estimation de p .
- Les bornes de Chernoff disent que

$$\Pr(X/n > p + \delta) = \Pr(X > np + n\delta) = \Pr(X > np(1 + \delta/p)) \leq e^{-np\delta^2/(3p^2)}$$

$$\text{Soit } \Pr(X/n > p + \delta) \leq e^{-n\delta^2/(3p)}$$

Preuve

- On considère $X = \sum_i X_i$, avec $\Pr(X_i = 1) = p$, $\Pr(X_i = 0) = 1 - p$.
- Sur n expériences, on aura $E[X] = np$, et donc on va considérer $\bar{p} = X/n$ comme estimation de p .
- Les bornes de Chernoff disent que

$$\Pr(X/n > p + \delta) = \Pr(X > np + n\delta) = \Pr(X > np(1 + \delta/p)) \leq e^{-np\delta^2/(3p^2)}$$

$$\text{Soit } \Pr(X/n > p + \delta) \leq e^{-n\delta^2/(3p)}$$

- Symétriquement $\Pr(X/n < p - \delta) \leq e^{-n\delta^2/(2p)}$

Preuve

- On considère $X = \sum_i X_i$, avec $\Pr(X_i = 1) = p$, $\Pr(X_i = 0) = 1 - p$.
- Sur n expériences, on aura $E[X] = np$, et donc on va considérer $\bar{p} = X/n$ comme estimation de p .
- Les bornes de Chernoff disent que

$$\Pr(X/n > p + \delta) = \Pr(X > np + n\delta) = \Pr(X > np(1 + \delta/p)) \leq e^{-np\delta^2/(3p^2)}$$

$$\text{Soit } \Pr(X/n > p + \delta) \leq e^{-n\delta^2/(3p)}$$

- Symétriquement $\Pr(X/n < p - \delta) \leq e^{-n\delta^2/(2p)}$
- On ne connaît pas p , mais on sait qu'il est plus petit que 1.

Preuve

- On considère $X = \sum_i X_i$, avec $\Pr(X_i = 1) = p$, $\Pr(X_i = 0) = 1 - p$.
- Sur n expériences, on aura $E[X] = np$, et donc on va considérer $\bar{p} = X/n$ comme estimation de p .
- Les bornes de Chernoff disent que

$$\Pr(X/n > p + \delta) = \Pr(X > np + n\delta) = \Pr(X > np(1 + \delta/p)) \leq e^{-np\delta^2/(3p^2)}$$

$$\text{Soit } \Pr(X/n > p + \delta) \leq e^{-n\delta^2/(3p)}$$

- Symétriquement $\Pr(X/n < p - \delta) \leq e^{-n\delta^2/(2p)}$
- On ne connaît pas p , mais on sait qu'il est plus petit que 1.
- Donc

$$\Pr(X/n > p + \delta) \leq e^{-n\delta^2/(3)}$$

$$\Pr(X/n < p - \delta) \leq e^{-n\delta^2/(2)}$$

Preuve

- On considère $X = \sum_i X_i$, avec $\Pr(X_i = 1) = p$, $\Pr(X_i = 0) = 1 - p$.
- Sur n expériences, on aura $E[X] = np$, et donc on va considérer $\bar{p} = X/n$ comme estimation de p .
- Les bornes de Chernoff disent que

$$\Pr(X/n > p + \delta) = \Pr(X > np + n\delta) = \Pr(X > np(1 + \delta/p)) \leq e^{-np\delta^2/(3p^2)}$$

$$\text{Soit } \Pr(X/n > p + \delta) \leq e^{-n\delta^2/(3p)}$$

- Symétriquement $\Pr(X/n < p - \delta) \leq e^{-n\delta^2/(2p)}$
- On ne connaît pas p , mais on sait qu'il est plus petit que 1.
- Donc

$$\Pr(X/n > p + \delta) \leq e^{-n\delta^2/(3)}$$

$$\Pr(X/n < p - \delta) \leq e^{-n\delta^2/(2)}$$

- Soit, par un union bound,

$$\Pr(p \notin [X/n - \delta, X/n + \delta]) \leq 2e^{-n\delta^2/3}.$$

Preuve

- On considère $X = \sum_i X_i$, avec $\Pr(X_i = 1) = p$, $\Pr(X_i = 0) = 1 - p$.
- Sur n expériences, on aura $E[X] = np$, et donc on va considérer $\bar{p} = X/n$ comme estimation de p .
- Les bornes de Chernoff disent que

$$\Pr(X/n > p + \delta) = \Pr(X > np + n\delta) = \Pr(X > np(1 + \delta/p)) \leq e^{-np\delta^2/(3p^2)}$$

$$\text{Soit } \Pr(X/n > p + \delta) \leq e^{-n\delta^2/(3p)}$$

- Symétriquement $\Pr(X/n < p - \delta) \leq e^{-n\delta^2/(2p)}$
- On ne connaît pas p , mais on sait qu'il est plus petit que 1.
- Donc

$$\Pr(X/n > p + \delta) \leq e^{-n\delta^2/(3)}$$

$$\Pr(X/n < p - \delta) \leq e^{-n\delta^2/(2)}$$

- Soit, par un union bound,

$$\Pr(p \notin [X/n - \delta, X/n + \delta]) \leq 2e^{-n\delta^2/3}.$$

- Il suffit de prendre

$$n \geq \lceil 3 \frac{1}{\delta^2} \ln \frac{2}{\gamma} \rceil,$$

pour que cette quantité soit inférieure à γ .

Au menu

Retours sur quelques algorithmes

Classes probabilistes

Quelques résultats

Sous menu

Classes probabilistes

- Notion d'algorithme probabiliste

- Classe PP

- Classe BPP

- Classes RP et ZPP

- Relations entre classes

- Résumé

Notion d'algorithme probabiliste

- On peut voir un algorithme probabiliste comme un algorithme classique, si ce n'est qu'à certaines étapes, il est autorisé à lancer à tirer une pièce (non-biaisée) à pile ou face, et à prendre une décision basée sur le résultat.

Notion d'algorithme probabiliste

- On peut voir un algorithme probabiliste comme un algorithme classique, si ce n'est qu'à certaines étapes, il est autorisé à lancer à tirer une pièce (non-biaisée) à pile ou face, et à prendre une décision basée sur le résultat.
- Formellement,
 - ▶ on peut considérer qu'un algorithme probabiliste A qui travaille sur l'entrée w prend en fait deux entrées :
 1. d'une part, w , l'entrée,
 2. et, d'autre part, un suite (un mot) $y \in \{0, 1\}^*$ qui donne le résultat des tirages aléatoires successifs utilisés.

- On note $A(w, y)$ le résultat de l'algorithme sur l'entrée w avec les tirages aléatoires y .
- On ne s'intéressera dans la suite essentiellement qu'à des problèmes de décision :
 - ▶ $A(w, y)$ sera donc toujours soit le résultat "accepter" soit "rejeter".



$$\Pr(A \text{ accepte } w) = \frac{|\{y \in \{0, 1\}^k \mid A(w, y) \text{ accepte}\}|}{2^k},$$

Classes probabilistes

Notion d'algorithme probabiliste

Classe PP

Classe BPP

Classes RP et ZPP

Relations entre classes

Résumé

Definition

Un langage L est dans PP s'il existe un algorithme probabiliste A qui fonctionne en temps polynomial tel que pour tout mot w ,

1. si $w \in L$, $\Pr(A \text{ accepte } w) > 1/2$;
2. si $w \notin L$, $\Pr(A \text{ accepte } w) \leq 1/2$.

Théorème

On a

$$\text{NP} \subset \text{PP} \subset \text{PSPACE}.$$

Preuve

- Un mot w est dans un langage L de NP si et seulement s'il existe un certificat $u \in M^*$ tel que $\langle w, u \rangle \in A$, pour A un langage reconnaissable en temps polynomial par un certain algorithme A'
- Considérons un algorithme probabiliste B qui est obtenu à partir de cet algorithme A' , et qui se comporte de la façon suivante :
 - ▶ sur l'entrée w , l'algorithme tire à pile ou face.
 - ▶ S'il tire pile, alors il accepte.
 - ▶ Sinon, il simule l'algorithme A' sur w , en remplaçant chaque choix non-déterministe par un tirage à pile ou face.
- On a $\Pr(B \text{ accepte } w) = 1/2$ si $w \notin L$.
- On a $\Pr(B \text{ accepte } w) > 1/2$ pour $w \in L$.

Preuve

- Prouvons maintenant que $PP \subset PSPACE$:
 - ▶ Soit A l'algorithme probabiliste qui accepte $L \in PP$.
 - ▶ Il suffit de considérer l'algorithme B qui simule A sur tous les tirages possibles y des choix aléatoires, et compte le nombre de tirages y tels que $A(w, y)$ accepte, et qui accepte si ce nombre est supérieur à $1/2$.

Proposition

La classe PP est close par passage au complémentaire.

Preuve

- Il suffit de montrer que tout langage L de PP est reconnu par un algorithme probabiliste tel que la probabilité d'acceptation n'est jamais exactement $1/2$.
- En effet, il suffit alors d'inverser l'état d'acceptation et de rejet pour obtenir le résultat.
- Soit A un algorithme probabiliste qui fonctionne en temps $p(n)$ pour un polynôme $p(n)$, avec $p(n) > 1$.
- On construit un algorithme probabiliste qui sur une entrée w simule d'abord A pendant $p(n)$ étapes, et fait ensuite $p(n)$ tirages à pile ou face supplémentaires :
 - ▶ Il accepte si A a accepté, et si l'un de ces tirages a produit pile.
 - ▶ Sinon, il rejette.

Preuve

- Il suffit de montrer que tout langage L de PP est reconnu par un algorithme probabiliste tel que la probabilité d'acceptation n'est jamais exactement $1/2$.
- En effet, il suffit alors d'inverser l'état d'acceptation et de rejet pour obtenir le résultat.
- Soit A un algorithme probabiliste qui fonctionne en temps $p(n)$ pour un polynôme $p(n)$, avec $p(n) > 1$.
- On construit un algorithme probabiliste qui sur une entrée w simule d'abord A pendant $p(n)$ étapes, et fait ensuite $p(n)$ tirages à pile ou face supplémentaires :
 - ▶ Il accepte si A a accepté, et si l'un de ces tirages a produit pile.
 - ▶ Sinon, il rejette.
- Cette opération multiplie la probabilité d'acceptation de A par $2^{p(n)} - 1$, et permet de satisfaire la propriété.

Corollary

$$\text{NP} \cup \text{coNP} \subset \text{PP}.$$

Classes probabilistes

Notion d'algorithme probabiliste

Classe PP

Classe BPP

Classes RP et ZPP

Relations entre classes

Résumé

Classe BPP

Definition

Un langage L est dans BPP s'il existe un $0 < \epsilon < 1/2$, et un algorithme probabiliste qui fonctionne en temps polynomial tels que pour toute entrée w ,

1. si $w \in L$, $\Pr(A \text{ accepte } w) \geq \frac{1}{2} + \epsilon$;
2. si $w \notin L$, $\Pr(A \text{ rejette } w) \geq \frac{1}{2} + \epsilon$.

Autre formulation

Autrement dit, il existe une constante $0 < \epsilon'$ telle que pour toute entrée w ,

$$\Pr(A \text{ se trompe sur } w) \leq \epsilon'.$$

Réduction de l'erreur

- On peut fixer l'erreur à $\epsilon' = 1/4$ (ou toute autre constante entre 0 et 1/2 strictement).

- Proposition

un langage L est dans BPP s'il existe un algorithme probabiliste qui fonctionne en temps polynomial tel que pour toute entrée w ,

$$\Pr(A \text{ se trompe sur } w) \leq 1/4.$$

Preuve

- Il nous faut montrer qu'avec une erreur ϵ' , on peut rendre l'erreur inférieure à ϵ'' pour tout $0 < \epsilon'' < 1$.
- Soit L dans BPP reconnu par l'algorithme A . Soit k un entier. On considère l'algorithme B qui sur une entrée w de longueur n simule successivement et de façon indépendante $2k$ fois l'algorithme A . B accepte l'entrée w si majorité des réponses de chacune des simulations accepte, et refuse sinon.
- La probabilité que B se trompe est la probabilité qu'au moins k simulations parmi les $2k$ simulations de A sur l'entrée w donne une réponse erronée. La probabilité de cet évènement est donnée par

$$\sum_{i=k}^{2k} \Pr[A \text{ se trompe exactement } i \text{ fois sur les } 2k \text{ simulations}] = \sum_{i=k}^{2k} C_{2k}^i \delta^i (1-\delta)^{2k-i},$$

où δ est la probabilité d'erreur de A sur w , que l'on sait telle que $\delta \leq \epsilon' < 1/2$.

- Puisque $\delta/(1-\delta) < 1$ lorsque $\delta < 1/2$, aucun terme de cette somme ne diminue si l'on fixe $i = k$. On peut donc borner cette probabilité par

$$(k+1)C_{2k}^k \delta^k (1-\delta)^k \leq (k+1)2^{2k} \delta^k (1-\delta)^k \leq (k+1)(4\delta(1-\delta))^k.$$

- On peut majorer cela par $(k+1)(4\epsilon'(1-\epsilon'))^k$.
- Il suffit de choisir k assez grand pour que cette expression soit plus petite que l'erreur désirée.

- La même preuve montre qu'on peut en fait aussi écrire :

- **Proposition**

Dans la définition précédente de BPP, on peut aussi imposer que l'erreur soit au plus $(\frac{1}{2})^{p(|w|)}$ pour n'importe quel polynôme p sans changer la classe.

- En inversant les états d'acceptation et de rejet, on a clairement :

- Proposition

La classe BPP est close par passage au complémentaire.

Sous menu

Classes probabilistes

Notion d'algorithme probabiliste

Classe PP

Classe BPP

Classes RP et ZPP

Relations entre classes

Résumé

Classes RP et coRP

Definition

Un langage L est dans RP s'il existe un algorithme probabiliste A qui fonctionne en temps polynômial telle que pour toute entrée w

1. si $w \in L$, $\Pr(A \text{ se trompe sur } w) < 1/2$;
2. si $w \notin L$, $\Pr(A \text{ se trompe sur } w) = 0$.

Definition

Un langage L est dans coRP s'il existe un algorithme probabiliste qui fonctionne en temps polynômial telle que pour toute entrée w

1. si $w \in L$, $\Pr(A \text{ se trompe sur } w) = 0$;
2. si $w \notin L$, $\Pr(A \text{ se trompe sur } w) < 1/2$.

Théorème

Dans les définitions de RP et coRP on peut remplacer $1/2$ par n'importe quelle constante $0 < \epsilon \leq 1/2$ sans changer la classe.

Preuve

- Soit L dans RP reconnu par l'algorithme A .
- Soit k un entier.
- Considérons l'algorithme B qui sur une entrée w simule k fois A et accepte si et seulement si au moins l'une des k simulations de A accepte.
- L'algorithme B vérifie $\Pr(B \text{ rejette } w) \leq 1/2^k$ pour $w \in L$, et $\Pr(B \text{ accepte } w) = 0$ pour $w \notin L$.
- Il suffit de choisir k tel que $1/2^k < \epsilon$.
- La preuve pour coRP est symétrique.

Proposition

Dans la définition précédente, on peut imposer que l'erreur soit au plus $(\frac{1}{2})^{p(|w|)}$ pour n'importe quel polynôme p sans changer la classe : par exemple, pour RP , pour un polynôme p ,

1. *si $w \in L$, $\Pr(A \text{ se trompe sur } w) < (1/2)^{p(|w|)}$;*
2. *si $w \notin L$, $\Pr(A \text{ se trompe sur } w) = 0$.*

Definition

On définit

$$\text{ZPP} = \text{RP} \cap \text{coRP}.$$

ZPP et Algorithmes de Las Vegas

Théorème

ZPP correspond aux problèmes qui sont reconnus par un algorithme qui termine toujours avec une réponse correcte et qui termine en temps moyen polynomial en la taille de l'entrée.

Preuve

Sens \Rightarrow :

- soit $L \in \text{ZPP}$. Soient A et A' les algorithmes qui attestent que L est respectivement dans RP et dans coRP .
- Sur une entrée w , avec une probabilité $> 1/2$, A et A' donnent la même réponse.
 - ▶ Lorsque cela se produit, cette réponse est nécessairement correcte.
- Considérer l'algorithme A^* qui simule de façon alternée A et A' sur son entrée, et recommence jusqu'à ce que les deux machines soient d'accord.
 - ▶ A^* donne toujours une réponse correcte.
 - ▶ La moyenne du temps de réponse est plus faible que

$$(\text{temps}(A) + \text{temps}(A'))(1 * 1/2 + 2 * 1/4 + 3 * 1/8 + \dots)$$

soit

$$\mathcal{O}(\text{temps}(A) + \text{temps}(A')).$$

Preuve

Sens \Leftarrow :

- Soit t le temps d'exécution de l'algorithme probabiliste qui reconnaît le langage L .

- Soit A l'algorithme qui accepte un entrée w si et seulement si l'algorithme randomisé accepte l'entrée w en un temps $\leq 3 * t$, rejette sinon.
 - ▶ Pour $w \notin L$, l'algorithme A rejette toujours.

 - ▶ Pour $w \in L$, par l'inégalité de Markov, avec une probabilité $\geq 2/3$, l'algorithme A a le temps de simuler complètement l'algorithme et donc ne se trompe pas.

 - ▶ L est donc dans RP.
- L est dans coRP par l'argument symétrique.

- Par définition, puisque le complémentaire de RP et coRP et réciproquement :

- Proposition

ZPP est close par passage au complémentaire.

Sous menu

Classes probabilistes

Notion d'algorithme probabiliste

Classe PP

Classe BPP

Classes RP et ZPP

Relations entre classes

Résumé

Théorème

$P \subset RP$ et $P \subset coRP$.

Théorème

$P \subset RP$ et $P \subset coRP$.

Un algorithme non probabiliste est un algorithme probabiliste particulier.

Théorème

$P \subset RP$ et $P \subset \text{coRP}$.

Un algorithme non probabiliste est un algorithme probabiliste particulier.

Corollary

$P \subset \text{ZPP}$.

Par définitions :

Théorème

$\text{RP} \subset \text{BPP}$, $\text{coRP} \subset \text{BPP}$, $\text{BPP} \subset \text{PP}$.

Théorème

$\text{RP} \subset \text{NP}$, $\text{coRP} \subset \text{coNP}$.

Théorème

$\text{RP} \subset \text{NP}$, $\text{coRP} \subset \text{coNP}$.

Considérer l'algorithme probabiliste comme un algorithme non-déterministe : la suite des tirages aléatoires correspond à un certificat.

Théorème

$\text{RP} \subset \text{NP}$, $\text{coRP} \subset \text{coNP}$.

Considérer l'algorithme probabiliste comme un algorithme non-déterministe : la suite des tirages aléatoires correspond à un certificat.

Corollary

$\text{ZPP} \subset \text{NP}$ *et* $\text{ZPP} \subset \text{coNP}$.

Sous menu

Classes probabilistes

Notion d'algorithme probabiliste

Classe PP

Classe BPP

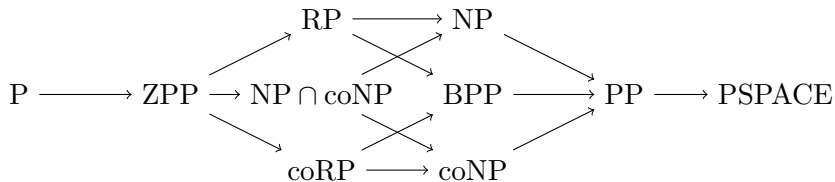
Classes RP et ZPP

Relations entre classes

Résumé

Résumé

Chaque flèche correspond à une inclusion :



Au menu

Retours sur quelques algorithmes

Classes probabilistes

Quelques résultats

Sous menu

Quelques résultats

Sur l'utilisation de pièces biaisées

BPP et circuits polynomiaux

Sur l'utilisation de pièces biaisées

- Notre modèle d'algorithme probabiliste suppose l'existence d'une opération qui permet de tirer une pièce non-biaisée à pile ou face.
- Que se passerait-il s'il on utilisait une pièce biaisée, c'est-à-dire une pièce qui serait pile avec probabilité ρ et face avec probabilité $1 - \rho$, avec $\rho \neq 1/2$?
- Obtiendrait-on plus de puissance ?

Théorème

- *On peut simuler un algorithme qui utilise une pièce non-biaisée par un algorithme qui utilise une pièce biaisée.*
- *Cela introduit un surcoût de temps dont la moyenne est en $\mathcal{O}\left(\frac{1}{\rho(1-\rho)}\right)$.*

Preuve

- On construit un algorithme qui utilise une pièce biaisée pour simuler chaque tirage d'une pièce non-biaisée :

Preuve

- On construit un algorithme qui utilise une pièce biaisée pour simuler chaque tirage d'une pièce non-biaisée :
 - ▶ pour simuler un tel tirage, l'algorithme tire une paire de pièce de biais ρ jusqu'à obtenir deux résultats différents sur ces pièces, c'est-à-dire face et pile, ou pile et face.

Preuve

- On construit un algorithme qui utilise une pièce biaisée pour simuler chaque tirage d'une pièce non-biaisée :
 - ▶ pour simuler un tel tirage, l'algorithme tire une paire de pièce de biais ρ jusqu'à obtenir deux résultats différents sur ces pièces, c'est-à-dire face et pile, ou pile et face.
 - ▶ A ce moment là, si la première pièce est pile, l'algorithme répond pile, et si la première pièce est face, l'algorithme répond face.

Preuve

- On construit un algorithme qui utilise une pièce biaisée pour simuler chaque tirage d'une pièce non-biaisée :
 - ▶ pour simuler un tel tirage, l'algorithme tire une paire de pièce de biais ρ jusqu'à obtenir deux résultats différents sur ces pièces, c'est-à-dire face et pile, ou pile et face.
 - ▶ A ce moment là, si la première pièce est pile, l'algorithme répond pile, et si la première pièce est face, l'algorithme répond face.
- La probabilité qu'une paire de pièces biaisées fassent apparaître face et pile est $\rho(1 - \rho)$, alors que la probabilité de faire apparaître pile et face est $\rho(1 - \rho)$.

Preuve

- On construit un algorithme qui utilise une pièce biaisée pour simuler chaque tirage d'une pièce non-biaisée :
 - ▶ pour simuler un tel tirage, l'algorithme tire une paire de pièce de biais ρ jusqu'à obtenir deux résultats différents sur ces pièces, c'est-à-dire face et pile, ou pile et face.
 - ▶ A ce moment là, si la première pièce est pile, l'algorithme répond pile, et si la première pièce est face, l'algorithme répond face.
- La probabilité qu'une paire de pièces biaisées fassent apparaître face et pile est $\rho(1 - \rho)$, alors que la probabilité de faire apparaître pile et face est $\rho(1 - \rho)$.
- Par conséquent, chaque étape permet de répondre avec la probabilité $2\rho(1 - \rho)$.

Preuve

- On construit un algorithme qui utilise une pièce biaisée pour simuler chaque tirage d'une pièce non-biaisée :
 - ▶ pour simuler un tel tirage, l'algorithme tire une paire de pièce de biais ρ jusqu'à obtenir deux résultats différents sur ces pièces, c'est-à-dire face et pile, ou pile et face.
 - ▶ A ce moment là, si la première pièce est pile, l'algorithme répond pile, et si la première pièce est face, l'algorithme répond face.
- La probabilité qu'une paire de pièces biaisées fassent apparaître face et pile est $\rho(1 - \rho)$, alors que la probabilité de faire apparaître pile et face est $\rho(1 - \rho)$.
- Par conséquent, chaque étape permet de répondre avec la probabilité $2\rho(1 - \rho)$.
- En temps moyen $\mathcal{O}\left(\frac{1}{\rho(1-\rho)}\right)$, on finira donc pas répondre.

Preuve

- On construit un algorithme qui utilise une pièce biaisée pour simuler chaque tirage d'une pièce non-biaisée :
 - ▶ pour simuler un tel tirage, l'algorithme tire une paire de pièce de biais ρ jusqu'à obtenir deux résultats différents sur ces pièces, c'est-à-dire face et pile, ou pile et face.
 - ▶ A ce moment là, si la première pièce est pile, l'algorithme répond pile, et si la première pièce est face, l'algorithme répond face.
- La probabilité qu'une paire de pièces biaisées fassent apparaître face et pile est $\rho(1 - \rho)$, alors que la probabilité de faire apparaître pile et face est $\rho(1 - \rho)$.
- Par conséquent, chaque étape permet de répondre avec la probabilité $2\rho(1 - \rho)$.
- En temps moyen $\mathcal{O}\left(\frac{1}{\rho(1-\rho)}\right)$, on finira donc pas répondre.
- Si l'on conditionne sur le fait que l'un de ces deux événements s'est produit à une étape donnée, la probabilité de répondre pile ou face est bien équiprobable.

Théorème

- *Supposons que*

1. *l'écriture en binaire de ρ soit*

$$\rho = 0.p_1p_2 \dots ,$$

2. *et telle que l'on puisse produire le i ème bit de ρ en temps polynomial en i .*

- *Alors, on peut simuler un algorithme avec une pile de biais ρ par un algorithme probabiliste au sens précédent (sans biais) en temps moyen $\mathcal{O}(1)$.*

Preuve

- L'algorithme produit par tirages non biaisés une suite de bits aléatoires b_1, b_2, \dots , un à un, où le bit b_i est produit à l'étape i .

Preuve

- L'algorithme produit par tirages non biaisés une suite de bits aléatoires b_1, b_2, \dots , un à un, où le bit b_i est produit à l'étape i .
 - ▶ Si $b_i < p_i$, alors l'algorithme produit la réponse pile, et s'arrête.

Preuve

- L'algorithme produit par tirages non biaisés une suite de bits aléatoires b_1, b_2, \dots , un à un, où le bit b_i est produit à l'étape i .
 - ▶ Si $b_i < p_i$, alors l'algorithme produit la réponse pile, et s'arrête.
 - ▶ Si $b_i > p_i$, la machine produit la réponse face et s'arrête.

Preuve

- L'algorithme produit par tirages non biaisés une suite de bits aléatoires b_1, b_2, \dots , un à un, où le bit b_i est produit à l'étape i .
 - ▶ Si $b_i < p_i$, alors l'algorithme produit la réponse pile, et s'arrête.
 - ▶ Si $b_i > p_i$, la machine produit la réponse face et s'arrête.
 - ▶ Sinon, l'algorithme passe à l'étape $i + 1$.

Preuve

- L'algorithme produit par tirages non biaisés une suite de bits aléatoires b_1, b_2, \dots , un à un, où le bit b_i est produit à l'étape i .
 - ▶ Si $b_i < p_i$, alors l'algorithme produit la réponse pile, et s'arrête.
 - ▶ Si $b_i > p_i$, la machine produit la réponse face et s'arrête.
 - ▶ Sinon, l'algorithme passe à l'étape $i + 1$.
- Clairement, l'algorithme atteint l'étape $i + 1$ si et seulement si $b_j = p_j$ pour tout $j \leq i$, ce qui se produit avec probabilité $1/2^i$.

Preuve

- L'algorithme produit par tirages non biaisés une suite de bits aléatoires b_1, b_2, \dots , un à un, où le bit b_i est produit à l'étape i .
 - ▶ Si $b_i < p_i$, alors l'algorithme produit la réponse pile, et s'arrête.
 - ▶ Si $b_i > p_i$, la machine produit la réponse face et s'arrête.
 - ▶ Sinon, l'algorithme passe à l'étape $i + 1$.
- Clairement, l'algorithme atteint l'étape $i + 1$ si et seulement si $b_j = p_j$ pour tout $j \leq i$, ce qui se produit avec probabilité $1/2^i$.
- La probabilité de répondre pile est donc $\sum_i p_i 1/2^i$, ce qui vaut exactement ρ .

Preuve

- L'algorithme produit par tirages non biaisés une suite de bits aléatoires b_1, b_2, \dots , un à un, où le bit b_i est produit à l'étape i .
 - ▶ Si $b_i < p_i$, alors l'algorithme produit la réponse pile, et s'arrête.
 - ▶ Si $b_i > p_i$, la machine produit la réponse face et s'arrête.
 - ▶ Sinon, l'algorithme passe à l'étape $i + 1$.
- Clairement, l'algorithme atteint l'étape $i + 1$ si et seulement si $b_j = p_j$ pour tout $j \leq i$, ce qui se produit avec probabilité $1/2^i$.
- La probabilité de répondre pile est donc $\sum_i p_i 1/2^i$, ce qui vaut exactement ρ .
- D'autre part, le temps moyen, de la forme $\sum_i i^c 1/2^i$ pour une constante c , est bien borné par une constante.

Sous menu

Quelques résultats

Sur l'utilisation de pièces biaisées

BPP et circuits polynomiaux

BPP et circuits polynomiaux

- Rappelons que \mathbb{P} désigne les langages reconnus par une famille de circuits de taille polynomiale.
- Théorème (Sipser-Gâcs)

$$\text{BPP} \subset \mathbb{P}.$$

Preuve

- Par les techniques de réduction d'erreur discutées auparavant, on sait que pour un langage L de BPP, on peut construire un algorithme A qui sur les entrées de taille n utilise m tirages aléatoires, et tel que pour tout w de longueur n , $\Pr(A \text{ se trompe sur } w) \leq 2^{-n-1}$.

Preuve

- Par les techniques de réduction d'erreur discutées auparavant, on sait que pour un langage L de BPP, on peut construire un algorithme A qui sur les entrées de taille n utilise m tirages aléatoires, et tel que pour tout w de longueur n , $\Pr(A \text{ se trompe sur } w) \leq 2^{-n-1}$.
- Qualifions un mot $y \in \{0,1\}^m$ (correspondant à une suite de tirages aléatoires) de *mauvais* si pour une entrée w , A se trompe sur une entrée w de longueur n , et de *bon* sinon.

Preuve

- Par les techniques de réduction d'erreur discutées auparavant, on sait que pour un langage L de BPP, on peut construire un algorithme A qui sur les entrées de taille n utilise m tirages aléatoires, et tel que pour tout w de longueur n , $\Pr(A \text{ se trompe sur } w) \leq 2^{-n-1}$.
- Qualifions un mot $y \in \{0,1\}^m$ (correspondant à une suite de tirages aléatoires) de *mauvais* si pour une entrée w , A se trompe sur une entrée w de longueur n , et de *bon* sinon.
- Pour toute entrée w , il y a au plus $\frac{2^m}{2^{n+1}}$ mots qui sont mauvais pour w .

Preuve

- Par les techniques de réduction d'erreur discutées auparavant, on sait que pour un langage L de BPP, on peut construire un algorithme A qui sur les entrées de taille n utilise m tirages aléatoires, et tel que pour tout w de longueur n , $\Pr(A \text{ se trompe sur } w) \leq 2^{-n-1}$.
- Qualifions un mot $y \in \{0,1\}^m$ (correspondant à une suite de tirages aléatoires) de *mauvais* si pour une entrée w , A se trompe sur une entrée w de longueur n , et de *bon* sinon.
- Pour toute entrée w , il y a au plus $\frac{2^m}{2^{n+1}}$ mots qui sont mauvais pour w .
- En additionnant sur tous les mots w de longueur n , il y a au plus $2^n \cdot \frac{2^m}{2^{n+1}} = 2^m/2$ mots qui sont mauvais pour *au moins un mot* w .

Preuve

- Par les techniques de réduction d'erreur discutées auparavant, on sait que pour un langage L de BPP, on peut construire un algorithme A qui sur les entrées de taille n utilise m tirages aléatoires, et tel que pour tout w de longueur n , $\Pr(A \text{ se trompe sur } w) \leq 2^{-n-1}$.
- Qualifions un mot $y \in \{0,1\}^m$ (correspondant à une suite de tirages aléatoires) de *mauvais* si pour une entrée w , A se trompe sur une entrée w de longueur n , et de *bon* sinon.
- Pour toute entrée w , il y a au plus $\frac{2^m}{2^{n+1}}$ mots qui sont mauvais pour w .
- En additionnant sur tous les mots w de longueur n , il y a au plus $2^n \cdot \frac{2^m}{2^{n+1}} = 2^m/2$ mots qui sont mauvais pour *au moins un* mot w .
- Il doit exister un mot y qui est bon pour *toute* entrée w de taille n .

Preuve

- Par les techniques de réduction d'erreur discutées auparavant, on sait que pour un langage L de BPP, on peut construire un algorithme A qui sur les entrées de taille n utilise m tirages aléatoires, et tel que pour tout w de longueur n , $\Pr(A \text{ se trompe sur } w) \leq 2^{-n-1}$.
- Qualifions un mot $y \in \{0,1\}^m$ (correspondant à une suite de tirages aléatoires) de *mauvais* si pour une entrée w , A se trompe sur une entrée w de longueur n , et de *bon* sinon.
- Pour toute entrée w , il y a au plus $\frac{2^m}{2^{n+1}}$ mots qui sont mauvais pour w .
- En additionnant sur tous les mots w de longueur n , il y a au plus $2^n \cdot \frac{2^m}{2^{n+1}} = 2^m/2$ mots qui sont mauvais pour *au moins un* mot w .
- Il doit exister un mot y qui est bon pour *toute* entrée w de taille n .
- On peut alors construire pour chaque taille n d'entrée un circuit pour les mots de taille n qui contient codé "en dur" dans le circuit ce mot y .