Algorithmique Probabiliste

Philippe Duchon

LaBRI - ENSEIRB-Matméca - Université de Bordeaux

2012-13

The story so far...

On a vu deux exemples d'algorithmes, avec des propriétés bien différentes :

- **RandQuickSort** est un algorithme randomisé dont le résultat est déterministe ; c'est son *temps d'exécution* qui est aléatoire (et il est intéressant pour “battre” de mauvaises instances)

The story so far...

On a vu deux exemples d'algorithmes, avec des propriétés bien différentes :

- **RandQuickSort** est un algorithme randomisé dont le résultat est déterministe ; c'est son *temps d'exécution* qui est aléatoire (et il est intéressant pour "battre" de mauvaises instances)
- Le **test d'égalité de deux chaînes** est un algorithme dont le *résultat* est lui-même aléatoire, il peut même donner un résultat **faux** ; on s'est attaché, en premier lieu, à **borner** (pour toute instance) la probabilité que le résultat soit faux.

The story so far…

On a vu deux exemples d'algorithmes, avec des propriétés bien différentes :

- **RandQuickSort** est un algorithme randomisé dont le résultat est déterministe ; c'est son *temps d'exécution* qui est aléatoire (et il est intéressant pour "battre" de mauvaises instances)
- Le **test d'égalité de deux chaînes** est un algorithme dont le *résultat* est lui-même aléatoire, il peut même donner un résultat **faux** ; on s'est attaché, en premier lieu, à **borner** (pour toute instance) la probabilité que le résultat soit faux.
- Dans le cas du test d'égalité, le caractère **probabiliste** de l'algorithme (et le fait que des exécutions successives soient considérées comme indépendantes) fait qu'il devient *utile* de **répéter le même algorithme plusieurs fois avec les mêmes données** – on s'en sert pour améliorer la probabilité d'obtenir un résultat correct.

Las Vegas vs Monte Carlo

- Un algorithme **Las Vegas** est un algorithme qui, pour toute instance,
  - s'arrête avec probabilité 1
  - donne un résultat correct avec probabilité 1

Las Vegas vs Monte Carlo

- Un algorithme **Las Vegas** est un algorithme qui, pour toute instance,
  - s'arrête avec probabilité 1
  - donne un résultat correct avec probabilité 1
- Un algorithme **Monte Carlo**, de **probabilité d'erreur** $\leq p$, est un algorithme qui, pour toute instance x,
  - s'arrête avec probabilité 1
  - donne un résultat correct avec probabilité au moins $1 - p$

  (éventuellement, p peut être exprimée comme une fonction de la taille de x : par exemple, $p = 2/|x|^2$; en tout cas, cette borne doit être valable pour **toute** instance de la taille en question)

Monte Carlo : 1MC vs 2MC

Parmi les algorithmes Monte Carlo pour les **problèmes de décision** (dont la réponse $f(x)$, pour toute instance, est `Oui` ou `Non`), on distingue deux cas :

- Un algorithme $\mathcal{A}$ est **2MC** (Monte Carlo à erreur bilatérale), par défaut : pour toute instance x, $\mathbb{P}(\mathcal{A}(x) = f(x)) \geq 1 - p$

Monte Carlo : 1MC vs 2MC

Parmi les algorithmes Monte Carlo pour les **problèmes de décision** (dont la réponse $f(x)$, pour toute instance, est `Oui` ou `Non`), on distingue deux cas :

- Un algorithme $\mathcal{A}$ est **2MC** (Monte Carlo à erreur bilatérale), par défaut : pour toute instance x, $\mathbb{P}(\mathcal{A}(x) = f(x)) \geq 1 - p$
- Un algorithme $\mathcal{A}$ est **1MC** (Monte Carlo à erreur unilatérale) s'il satisfait les deux contraintes :
  - pour toute **instance positive** (x tel que $f(x) = \texttt{Oui}$), $\mathcal{A}$ donne une réponse correcte avec probabilité au moins $1 - p$: $\mathbb{P}(\mathcal{A}(x) = \texttt{Oui}) \geq 1 - p$;
  - pour toute **instance négative** (x tel que $f(x) = \texttt{Non}$), $\mathcal{A}$ donne **toujours** une réponse correcte : $\mathbb{P}(\mathcal{A}(x) = \texttt{Non}) = 1$.

Monte Carlo : 1MC vs 2MC

Parmi les algorithmes Monte Carlo pour les **problèmes de décision** (dont la réponse $f(x)$, pour toute instance, est `Oui` ou `Non`), on distingue deux cas :

- Un algorithme $\mathcal{A}$ est **2MC** (Monte Carlo à erreur bilatérale), par défaut : pour toute instance x, $\mathbb{P}(\mathcal{A}(x) = f(x)) \geq 1 - p$
- Un algorithme $\mathcal{A}$ est **1MC** (Monte Carlo à erreur unilatérale) s'il satisfait les deux contraintes :
  - pour toute **instance positive** (x tel que $f(x) = \texttt{Oui}$), $\mathcal{A}$ donne une réponse correcte avec probabilité au moins $1 - p$: $\mathbb{P}(\mathcal{A}(x) = \texttt{Oui}) \geq 1 - p$;
  - pour toute **instance négative** (x tel que $f(x) = \texttt{Non}$), $\mathcal{A}$ donne **toujours** une réponse correcte : $\mathbb{P}(\mathcal{A}(x) = \texttt{Non}) = 1$.
- (Un algorithme 1MC ne peut se tromper que dans un seul sens : dans le doute, il **doit** répondre `Non`)

Exemples de M. Jourdain

- **RandQuickSort** est un algorithme **Las Vegas** (pour le tri)

Exemples de M. Jourdain

- **RandQuickSort** est un algorithme **Las Vegas** (pour le tri)
- Un algorithme **2MC** de probabilité d'erreur $1/2$ n'a aucune utilité (même pour un problème indécidable) (pourquoi ?)

Exemples de M. Jourdain

- **RandQuickSort** est un algorithme **Las Vegas** (pour le tri)
- Un algorithme **2MC** de probabilité d'erreur $1/2$ n'a aucune utilité (même pour un problème indécidable) (pourquoi ?)
- L'algorithme **TestÉgalité** est un algorithme **Monte Carlo**, de probabilité d'erreur $2\ln(2)/n$ (pour les chaînes de longueur n)

Exemples de M. Jourdain

- **RandQuickSort** est un algorithme **Las Vegas** (pour le tri)
- Un algorithme **2MC** de probabilité d'erreur $1/2$ n'a aucune utilité (même pour un problème indécidable) (pourquoi ?)
- L'algorithme **TestÉgalité** est un algorithme **Monte Carlo**, de probabilité d'erreur $2\ln(2)/n$ (pour les chaînes de longueur n)
- Pour peu qu'on considère le "bon" problème de décision ("Ces deux chaînes sont-elles *différentes*), il s'agit d'un algorithme **1MC** de probabilité d'erreur $2\ln(2)/n$ (algorithme **1MC*** : la probabilité d'erreur tend vers 0)

Monte Carlo : probabilité d'erreur

- Un algorithme **2MC** avec une probabilité d'erreur supérieure ou égale à $1/2$ n'a aucun intérêt

Monte Carlo : probabilité d'erreur

- Un algorithme **2MC** avec une probabilité d'erreur supérieure ou égale à $1/2$ n'a aucun intérêt
- Pour un algorithme **1MC**, toute probabilité d'erreur < 1 est pertinente

Monte Carlo : probabilité d'erreur

- Un algorithme **2MC** avec une probabilité d'erreur supérieure ou égale à $1/2$ n'a aucun intérêt
- Pour un algorithme **1MC**, toute probabilité d'erreur < 1 est pertinente
- En **répétant** un algorithme **1MC** k fois (et en retournant `Oui` si au moins une répétition retourne `Oui`), on fait passer la probabilité d'erreur de p à p^k

Monte Carlo : probabilité d'erreur

- Un algorithme **2MC** avec une probabilité d'erreur supérieure ou égale à $1/2$ n'a aucun intérêt
- Pour un algorithme **1MC**, toute probabilité d'erreur < 1 est pertinente
- En **répétant** un algorithme **1MC** k fois (et en retournant `Oui` si au moins une répétition retourne `Oui`), on fait passer la probabilité d'erreur de p à p^k
- En particulier :

Monte Carlo : probabilité d'erreur

- Un algorithme **2MC** avec une probabilité d'erreur supérieure ou égale à 1/2 n'a aucun intérêt
- Pour un algorithme **1MC**, toute probabilité d'erreur < 1 est pertinente
- En **répétant** un algorithme **1MC** k fois (et en retournant `Oui` si au moins une répétition retourne `Oui`), on fait passer la probabilité d'erreur de p à p^k
- En particulier :
  - On atteint $p' = \varepsilon$ pour $k = \lceil \log_{1/p}(1/\varepsilon) \rceil$

Monte Carlo : probabilité d'erreur

- Un algorithme **2MC** avec une probabilité d'erreur supérieure ou égale à $1/2$ n'a aucun intérêt
- Pour un algorithme **1MC**, toute probabilité d'erreur < 1 est pertinente
- En **répétant** un algorithme **1MC** k fois (et en retournant `Oui` si au moins une répétition retourne `Oui`), on fait passer la probabilité d'erreur de p à p^k
- En particulier :
  - On atteint $p' = \varepsilon$ pour $k = \lceil \log_{1/p}(1/\varepsilon) \rceil$
  - Si $p \leq 1 - c/n^\alpha$, on a $p' \leq \varepsilon$ dès que

$$k \geq \frac{n^\alpha}{c} \ln\left(\frac{1}{\varepsilon}\right)$$

Monte Carlo : probabilité d'erreur

- Un algorithme **2MC** avec une probabilité d'erreur supérieure ou égale à $1/2$ n'a aucun intérêt
- Pour un algorithme **1MC**, toute probabilité d'erreur < 1 est pertinente
- En **répétant** un algorithme **1MC** k fois (et en retournant `Oui` si au moins une répétition retourne `Oui`), on fait passer la probabilité d'erreur de p à p^k
- En particulier :
  - On atteint $p' = \varepsilon$ pour $k = \lceil \log_{1/p}(1/\varepsilon) \rceil$
  - Si $p \leq 1 - c/n^\alpha$, on a $p' \leq \varepsilon$ dès que

$$k \geq \frac{n^\alpha}{c} \ln\left(\frac{1}{\varepsilon}\right)$$

- On peut également (moins efficacement) diminuer par répétition la probabilité d'erreur d'un algorithme **2MC**

Monte Carlo : probabilité d'erreur

- Un algorithme **2MC** avec une probabilité d'erreur supérieure ou égale à $1/2$ n'a aucun intérêt
- Pour un algorithme **1MC**, toute probabilité d'erreur < 1 est pertinente
- En **répétant** un algorithme **1MC** k fois (et en retournant `Oui` si au moins une répétition retourne `Oui`), on fait passer la probabilité d'erreur de p à p^k
- En particulier :
  - On atteint $p' = \varepsilon$ pour $k = \lceil \log_{1/p}(1/\varepsilon) \rceil$
  - Si $p \leq 1 - c/n^\alpha$, on a $p' \leq \varepsilon$ dès que

$$k \geq \frac{n^\alpha}{c} \ln\left(\frac{1}{\varepsilon}\right)$$

- On peut également (moins efficacement) diminuer par répétition la probabilité d'erreur d'un algorithme **2MC**
- (Le temps d'exécution, dans le cas le pire comme en moyenne, est multiplié par k)

Temps d'exécution

Pour l'étude des complexités, on distingue deux façons de mesurer le temps d'exécution d'un algorithme (en raison du caractère *aléatoire* de ce temps) :

- **Cas le pire :** un algorithme est *à temps d'exécution polynomial* s'il existe une fonction f à croissance polynomiale ($f(n) = O(n^k)$ pour un certain k), telle que, pour toute instance x et toute exécution de l'algorithme sur x, le temps soit au plus $f(|x|)$.

Temps d'exécution

Pour l'étude des complexités, on distingue deux façons de mesurer le temps d'exécution d'un algorithme (en raison du caractère *aléatoire* de ce temps) :

- **Cas le pire :** un algorithme est *à temps d'exécution polynomial* s'il existe une fonction f à croissance polynomiale ($f(n) = O(n^k)$ pour un certain k), telle que, pour toute instance x et toute exécution de l'algorithme sur x, le temps soit au plus $f(|x|)$.
- **En moyenne (sur les exécutions) :** un algorithme $\mathcal{A}$ est *à temps moyen polynomial* s'il existe une fonction f à croissance polynomiale telle que, pour toute instance x, l'**espérance** du temps d'exécution de $\mathcal{A}$ sur l'entrée x est inférieure à $f(|x|)$

Temps d'exécution

Pour l'étude des complexités, on distingue deux façons de mesurer le temps d'exécution d'un algorithme (en raison du caractère *aléatoire* de ce temps) :

- **Cas le pire :** un algorithme est *à temps d'exécution polynomial* s'il existe une fonction f à croissance polynomiale ($f(n) = O(n^k)$ pour un certain k), telle que, pour toute instance x et toute exécution de l'algorithme sur x, le temps soit au plus $f(|x|)$.
- **En moyenne (sur les exécutions) :** un algorithme $\mathcal{A}$ est *à temps moyen polynomial* s'il existe une fonction f à croissance polynomiale telle que, pour toute instance x, l'**espérance** du temps d'exécution de $\mathcal{A}$ sur l'entrée x est inférieure à $f(|x|)$
- (il s'agit de "complexité en moyenne [sur les exécutions] dans le cas le pire [sur les entrées]")

Classes de complexité

- $\mathcal{RP}$ (Randomized Polynomial) : un problème de décision P est un problème $\mathcal{RP}$ s'il existe pour P un algorithme **1MC**, de probabilité d'erreur $1/2$, *à temps d'exécution polynomial*.

Classes de complexité

- $\mathcal{RP}$ (Randomized Polynomial) : un problème de décision P est un problème $\mathcal{RP}$ s'il existe pour P un algorithme **1MC**, de probabilité d'erreur $1/2$, *à temps d'exécution polynomial*.
- $\mathcal{ZPP}$ (Zero-Probability Polynomial) : un problème de décision P est un problème $\mathcal{ZPP}$ s'il existe pour P un algorithme **Las Vegas** *à temps moyen polynomial*.

Classes de complexité

- $\mathcal{RP}$ (Randomized Polynomial) : un problème de décision P est un problème $\mathcal{RP}$ s'il existe pour P un algorithme **1MC**, de probabilité d'erreur $1/2$, *à temps d'exécution polynomial*.
- $\mathcal{ZPP}$ (Zero-Probability Polynomial) : un problème de décision P est un problème $\mathcal{ZPP}$ s'il existe pour P un algorithme **Las Vegas** *à temps moyen polynomial*.
- co-$\mathcal{RP}$: un problème de décision P est un problème co-$\mathcal{RP}$ si le problème complémentaire (on échange les réponses `Oui` et `Non`) est dans $\mathcal{RP}$.

Classes de complexité

- $\mathcal{RP}$ (Randomized Polynomial) : un problème de décision P est un problème $\mathcal{RP}$ s'il existe pour P un algorithme **1MC**, de probabilité d'erreur $1/2$, *à temps d'exécution polynomial*.
- $\mathcal{ZPP}$ (Zero-Probability Polynomial) : un problème de décision P est un problème $\mathcal{ZPP}$ s'il existe pour P un algorithme **Las Vegas** *à temps moyen polynomial*.
- co-$\mathcal{RP}$: un problème de décision P est un problème co-$\mathcal{RP}$ si le problème complémentaire (on échange les réponses `Oui` et `Non`) est dans $\mathcal{RP}$.
- **Exercice :** montrer que l'on a

Classes de complexité

- $\mathcal{RP}$ (Randomized Polynomial) : un problème de décision P est un problème $\mathcal{RP}$ s'il existe pour P un algorithme **1MC**, de probabilité d'erreur $1/2$, *à temps d'exécution polynomial*.
- $\mathcal{ZPP}$ (Zero-Probability Polynomial) : un problème de décision P est un problème $\mathcal{ZPP}$ s'il existe pour P un algorithme **Las Vegas** *à temps moyen polynomial*.
- co-$\mathcal{RP}$: un problème de décision P est un problème co-$\mathcal{RP}$ si le problème complémentaire (on échange les réponses `Oui` et `Non`) est dans $\mathcal{RP}$.
- **Exercice :** montrer que l'on a
  - $\mathcal{ZPP} \subset \mathcal{RP}$

Classes de complexité

- $\mathcal{RP}$ (Randomized Polynomial) : un problème de décision P est un problème $\mathcal{RP}$ s'il existe pour P un algorithme **1MC**, de probabilité d'erreur $1/2$, *à temps d'exécution polynomial*.
- $\mathcal{ZPP}$ (Zero-Probability Polynomial) : un problème de décision P est un problème $\mathcal{ZPP}$ s'il existe pour P un algorithme **Las Vegas** *à temps moyen polynomial*.
- co-$\mathcal{RP}$: un problème de décision P est un problème co-$\mathcal{RP}$ si le problème complémentaire (on échange les réponses `Oui` et `Non`) est dans $\mathcal{RP}$.
- **Exercice :** montrer que l'on a
  - $\mathcal{ZPP} \subset \mathcal{RP}$
  - $\mathcal{ZPP} \subset$ co-$\mathcal{RP}$

Classes de complexité

- $\mathcal{RP}$ (Randomized Polynomial) : un problème de décision P est un problème $\mathcal{RP}$ s'il existe pour P un algorithme **1MC**, de probabilité d'erreur $1/2$, *à temps d'exécution polynomial*.
- $\mathcal{ZPP}$ (Zero-Probability Polynomial) : un problème de décision P est un problème $\mathcal{ZPP}$ s'il existe pour P un algorithme **Las Vegas** *à temps moyen polynomial*.
- co-$\mathcal{RP}$: un problème de décision P est un problème co-$\mathcal{RP}$ si le problème complémentaire (on échange les réponses `Oui` et `Non`) est dans $\mathcal{RP}$.
- **Exercice :** montrer que l'on a
  - $\mathcal{ZPP} \subset \mathcal{RP}$
  - $\mathcal{ZPP} \subset$ co-$\mathcal{RP}$
  - $\mathcal{RP} \cap$ co-$\mathcal{RP} \subset \mathcal{ZPP}$