# Classes de complexité probabilistes

Philippe Duchon

LaBRI

Aléa 2013

# Algorithmes et probabilités

Dans la communauté Aléa, on mélange souvent algorithmes et probabilités :

- **Analyse probabiliste :** on a un algorithme, et on essaie d'évaluer ses performances (en moyenne, en distribution) en supposant qu'il reçoit des données aléatoires (selon une distribution donnée - uniforme ou non)

# Algorithmes et probabilités

Dans la communauté Aléa, on mélange souvent algorithmes et probabilités :

- **Analyse probabiliste :** on a un algorithme, et on essaie d'évaluer ses performances (en moyenne, en distribution) en supposant qu'il reçoit des données aléatoires (selon une distribution donnée - uniforme ou non) *(pas le sujet)*

# Algorithmes et probabilités

Dans la communauté Aléa, on mélange souvent algorithmes et probabilités :

- **Analyse probabiliste :** on a un algorithme, et on essaie d'évaluer ses performances (en moyenne, en distribution) en supposant qu'il reçoit des données aléatoires (selon une distribution donnée - uniforme ou non) *(pas le sujet)*
- **Algorithmes probabilistes :** c'est l'algorithme à analyser qui a un comportement aléatoire (même sur des données parfaitement déterministes) : il "tire à pile ou face"

# Algorithmes et probabilités

Dans la communauté Aléa, on mélange souvent algorithmes et probabilités :

- **Analyse probabiliste :** on a un algorithme, et on essaie d'évaluer ses performances (en moyenne, en distribution) en supposant qu'il reçoit des données aléatoires (selon une distribution donnée - uniforme ou non) *(pas le sujet)*
- **Algorithmes probabilistes :** c'est l'algorithme à analyser qui a un comportement aléatoire (même sur des données parfaitement déterministes) : il "tire à pile ou face"
  - **Question centrale** (du point de vue de la théorie de la complexité) : "est-ce que ça aide" d'avoir la possibilité de tirer à pile ou face ?

# Algorithmes et probabilités

Dans la communauté Aléa, on mélange souvent algorithmes et probabilités :

- **Analyse probabiliste :** on a un algorithme, et on essaie d'évaluer ses performances (en moyenne, en distribution) en supposant qu'il reçoit des données aléatoires (selon une distribution donnée - uniforme ou non) *(pas le sujet)*
- **Algorithmes probabilistes :** c'est l'algorithme à analyser qui a un comportement aléatoire (même sur des données parfaitement déterministes) : il "tire à pile ou face"
  - **Question centrale** (du point de vue de la théorie de la complexité) : "est-ce que ça aide" d'avoir la possibilité de tirer à pile ou face ? (pas trop, on dirait)

# Algorithmes et probabilités

Dans la communauté Aléa, on mélange souvent algorithmes et probabilités :

- **Analyse probabiliste :** on a un algorithme, et on essaie d'évaluer ses performances (en moyenne, en distribution) en supposant qu'il reçoit des données aléatoires (selon une distribution donnée - uniforme ou non) *(pas le sujet)*
- **Algorithmes probabilistes :** c'est l'algorithme à analyser qui a un comportement aléatoire (même sur des données parfaitement déterministes) : il "tire à pile ou face"
  - **Question centrale** (du point de vue de la théorie de la complexité) : "est-ce que ça aide" d'avoir la possibilité de tirer à pile ou face ? (pas trop, on dirait)
  - **Du point de vue pratique :** l'utilisation de choix aléatoires donne accès à des algorithmes plus simples, et plus efficaces, que les algorithmes purement déterministes

# Algorithmes et probabilités

Dans la communauté Aléa, on mélange souvent algorithmes et probabilités :

- **Analyse probabiliste :** on a un algorithme, et on essaie d'évaluer ses performances (en moyenne, en distribution) en supposant qu'il reçoit des données aléatoires (selon une distribution donnée - uniforme ou non) *(pas le sujet)*
- **Algorithmes probabilistes :** c'est l'algorithme à analyser qui a un comportement aléatoire (même sur des données parfaitement déterministes) : il "tire à pile ou face"
  - **Question centrale** (du point de vue de la théorie de la complexité) : "est-ce que ça aide" d'avoir la possibilité de tirer à pile ou face ? (pas trop, on dirait)
  - **Du point de vue pratique :** l'utilisation de choix aléatoires donne accès à des algorithmes plus simples, et plus efficaces, que les algorithmes purement déterministes (mais ce n'est pas le sujet du jour)

# Au programme. . .

- **Première partie (mardi) :** quelques exemples de classes de complexité définies de manière probabiliste.
- **Seconde partie (jeudi) :** la question de la “simulabilité exacte” de certaines lois de probabilités.

# Complexité classique

- **Théorie de la complexité :** cherche à *classifier* les problèmes algorithmiques selon les “ressources” qu’il faut pour les résoudre

# Complexité classique

- **Théorie de la complexité :** cherche à *classifier* les problèmes algorithmiques selon les "ressources" qu'il faut pour les résoudre
- **"Ressources" :** temps de calcul, mémoire utilisée...

# Complexité classique

- **Théorie de la complexité :** cherche à *classifier* les problèmes algorithmiques selon les "ressources" qu'il faut pour les résoudre
- **"Ressources" :** temps de calcul, mémoire utilisée. . .
- **"Problème" :** une question qu'on se pose sur des données

# Complexité classique

- **Théorie de la complexité :** cherche à *classifier* les problèmes algorithmiques selon les "ressources" qu'il faut pour les résoudre
- **"Ressources" :** temps de calcul, mémoire utilisée. . .
- **"Problème" :** une question qu'on se pose sur des données
- **"Résoudre :"** un algorithme qui reçoit en entrée les données, et qui renvoie la réponse à la question

# Complexité classique

- **Théorie de la complexité :** cherche à *classifier* les problèmes algorithmiques selon les "ressources" qu'il faut pour les résoudre
- **"Ressources" :** temps de calcul, mémoire utilisée. . .
- **"Problème" :** une question qu'on se pose sur des données
- **"Résoudre :"** un algorithme qui reçoit en entrée les données, et qui renvoie la réponse à la question
- **Complexité (dans le cas le pire) d'un algorithme :** le maximum, sur *toutes les données possibles d'une longueur donnée n*, du temps de calcul (nombre d'opérations) que met l'algorithme pour calculer sa réponse

# Exemples de problèmes

- **Primalité :**
  - **Entrée :** un entier $m \in \mathbb{N}$
  - **Question :** est-ce que $m$ est premier ?

# Exemples de problèmes

- **Primalité :**
  - **Entrée :** un entier $m \in \mathbb{N}$
  - **Question :** est-ce que $m$ est premier ?
- **PetitDiviseur :**
  - **Entrée :** deux entiers positifs $m$ et $d$
  - **Question :** est-ce que $m$ a un diviseur $d'$, $1 < d' \leq d$ ?

# Exemples de problèmes

- **Primalité :**
  - **Entrée :** un entier $m \in \mathbb{N}$
  - **Question :** est-ce que $m$ est premier ?
- **PetitDiviseur :**
  - **Entrée :** deux entiers positifs $m$ et $d$
  - **Question :** est-ce que $m$ a un diviseur $d'$, $1 < d' \leq d$ ?
- **Tri :**
  - **Entrée :** une liste $q_1, \ldots, q_n$ de $n$ rationnels
  - **Sortie :** les mêmes $n$ nombres, dans l'ordre croissant

# Exemples de problèmes

- **Primalité :**
  - **Entrée :** un entier $m \in \mathbb{N}$
  - **Question :** est-ce que $m$ est premier ?
- **PetitDiviseur :**
  - **Entrée :** deux entiers positifs $m$ et $d$
  - **Question :** est-ce que $m$ a un diviseur $d'$, $1 < d' \leq d$ ?
- **Tri :**
  - **Entrée :** une liste $q_1, \ldots, q_n$ de $n$ rationnels
  - **Sortie :** les mêmes $n$ nombres, dans l'ordre croissant
- **Sélection :**
  - **Entrée :** une liste $q_1, \ldots, q_n$ de $n$ rationnels, un entier $1 \leq k \leq n$
  - **Sortie :** le $k$-ème plus petit des $q_i$

# Exemples de problèmes

- **Primalité :**
  - **Entrée :** un entier $m \in \mathbb{N}$
  - **Question :** est-ce que $m$ est premier ?
- **PetitDiviseur :**
  - **Entrée :** deux entiers positifs $m$ et $d$
  - **Question :** est-ce que $m$ a un diviseur $d'$, $1 < d' \leq d$ ?
- **Tri :**
  - **Entrée :** une liste $q_1, \ldots, q_n$ de $n$ rationnels
  - **Sortie :** les mêmes $n$ nombres, dans l'ordre croissant
- **Sélection :**
  - **Entrée :** une liste $q_1, \ldots, q_n$ de $n$ rationnels, un entier $1 \leq k \leq n$
  - **Sortie :** le $k$-ème plus petit des $q_i$
- **Problèmes de décision :** problèmes pour lesquels la réponse est toujours `Oui` ou `Non`

# Calcul déterministe, nondéterministe, probabiliste

- **Algorithme déterministe :** à chaque étape de calcul, l'algorithme a une unique action possible (arrêt compris)

# Calcul déterministe, nondéterministe, probabiliste

- **Algorithme déterministe :** à chaque étape de calcul, l'algorithme a une unique action possible (arrêt compris)
- **Algorithme nondéterministe :** à chaque étape, l'algorithme *peut* avoir soit une, soit deux actions possibles ; on ne précise pas "comment il en choisit une". L'algorithme peut avoir plusieurs exécutions différentes pour une même donnée $x$, il faut *préciser* ce qu'on considère comme "la" réponse de l'algorithme ; **exemple :** *si toutes les exécutions répondent* `Non`*, la réponse est* `Non` *; si au moins une exécution répond* `Oui`*, la réponse est* `Oui`*.*

# Calcul déterministe, nondéterministe, probabiliste

- **Algorithme déterministe :** à chaque étape de calcul, l'algorithme a une unique action possible (arrêt compris)
- **Algorithme nondéterministe :** à chaque étape, l'algorithme *peut* avoir soit une, soit deux actions possibles ; on ne précise pas "comment il en choisit une". L'algorithme peut avoir plusieurs exécutions différentes pour une même donnée $x$, il faut *préciser* ce qu'on considère comme "la" réponse de l'algorithme ; **exemple :** *si toutes les exécutions répondent* `Non`*, la réponse est* `Non` *; si au moins une exécution répond* `Oui`*, la réponse est* `Oui`*.*
- **Algorithme probabiliste :** l'algorithme est décrit comme un algorithme déterministe, mais a accès à une fonction `flip()`, *dont on suppose que chaque appel retourne un bit aléatoire : suite de variables aléatoires de Bernoulli de paramètre* $1/2$, ***indépendantes***.

# Les idées critiques

- En probabiliste, on peut tolérer une certaine probabilité d'obtenir un résultat incorrect ;

# Les idées critiques

- En probabiliste, on peut tolérer une certaine probabilité d'obtenir un résultat incorrect ;
- Contrairement au cas déterministe, ça peut valoir le coup d'exécuter *plusieurs fois* le même algorithme sur les *mêmes* données.

# Temps d'exécution

- **Notation :** pour un algorithme (déterministe ou probabiliste) $A$ et une donnée $x$, $T(A, x)$ désigne le temps d'exécution de $A$ sur $x$ (nombre d'étapes, éventuellement infini, du calcul) ; pour un algorithme nondéterministe, $T(A, x)$ désigne plutôt la longueur de la *plus longue* exécution de $A$ sur $x$.

# Temps d'exécution

- **Notation :** pour un algorithme (déterministe ou probabiliste) $A$ et une donnée $x$, $T(A, x)$ désigne le temps d'exécution de $A$ sur $x$ (nombre d'étapes, éventuellement infini, du calcul) ; pour un algorithme nondéterministe, $T(A, x)$ désigne plutôt la longueur de la *plus longue* exécution de $A$ sur $x$.
- **Notation :** $\ell(x)$ désigne la longueur de $x$ (considéré comme un mot fini sur un alphabet, typiquement binaire)

# Temps d'exécution

- **Notation :** pour un algorithme (déterministe ou probabiliste) $A$ et une donnée $x$, $T(A, x)$ désigne le temps d'exécution de $A$ sur $x$ (nombre d'étapes, éventuellement infini, du calcul) ; pour un algorithme nondéterministe, $T(A, x)$ désigne plutôt la longueur de la *plus longue* exécution de $A$ sur $x$.
- **Notation :** $\ell(x)$ désigne la longueur de $x$ (considéré comme un mot fini sur un alphabet, typiquement binaire)
- Un algorithme est *à temps borné par* $t$ ($t : \mathbb{N} \to \mathbb{N}$) si, pour *toute* instance $x$,

$$T(A, x) \leq t(\ell(x))$$

(avec probabilité 1, pour un algorithme probabiliste)

# Temps moyen

- Un algorithme (probabiliste) est *à temps moyen borné par $t$* si, pour toute instance $x$,

$$\mathbf{E}(T(A,x)) \leq t(\ell(x))$$

(l'espérance étant prise sur les exécutions de $A$ sur $x$)

# Temps moyen

- Un algorithme (probabiliste) est *à temps moyen borné par $t$* si, pour toute instance $x$,

$$\mathbf{E}(T(A,x)) \leq t(\ell(x))$$

(l'espérance étant prise sur les exécutions de $A$ sur $x$)

- Temps (moyen) polynomial : existence d'une fonction polynôme $t \in \mathbb{N}[X]$, telle que le temps (moyen) soit borné par $t$.

# Temps moyen

- Un algorithme (probabiliste) est *à temps moyen borné par $t$* si, pour toute instance $x$,

$$\mathbf{E}(T(A,x)) \leq t(\ell(x))$$

(l'espérance étant prise sur les exécutions de $A$ sur $x$)

- Temps (moyen) polynomial : existence d'une fonction polynôme $t \in \mathbb{N}[X]$, telle que le temps (moyen) soit borné par $t$.
- **Ça reste de l'analyse dans le cas le pire : on a des majorations [en loi, en espérance sur les exécutions] pour chaque instance, il n'y a aucune hypothèse probabiliste sur les instances**

# Probabilité d'erreur

- Plutôt que de définir "ce que calcule" un algorithme probabiliste, on lui définit une *probabilité d'erreur* pour le calcul d'une fonction $f$.

# Probabilité d'erreur

- Plutôt que de définir "ce que calcule" un algorithme probabiliste, on lui définit une *probabilité d'erreur* pour le calcul d'une fonction $f$.
- Pour une donnée $x$, la probabilité d'erreur de $A$ est

$$\mathbb{P}(A(x) \neq f(x));$$

# Probabilité d'erreur

- Plutôt que de définir "ce que calcule" un algorithme probabiliste, on lui définit une *probabilité d'erreur* pour le calcul d'une fonction $f$.
- Pour une donnée $x$, la probabilité d'erreur de $A$ est

$$\mathbb{P}(A(x) \neq f(x));$$

- L'algorithme est de probabilité d'erreur bornée par $e : \mathbb{N} \to [0, 1]$, si, pour toute donnée $x$,

$$\mathbb{P}(A(x) \neq f(x)) \leq e(\ell(x))$$

# Probabilité d'erreur

- Plutôt que de définir "ce que calcule" un algorithme probabiliste, on lui définit une *probabilité d'erreur* pour le calcul d'une fonction $f$.
- Pour une donnée $x$, la probabilité d'erreur de $A$ est

$$\mathbb{P}(A(x) \neq f(x));$$

- L'algorithme est de probabilité d'erreur bornée par $e : \mathbb{N} \to [0, 1]$, si, pour toute donnée $x$,

$$\mathbb{P}(A(x) \neq f(x)) \leq e(\ell(x))$$

- (Probabilité d'erreur *strictement* bornée par $e$ : inégalité stricte)

# P vs NP vs PP

- Un problème (de décision) est dans P (polynomial-time), s'il existe un algorithme (déterministe) à temps polynomial, qui, pour toute instance $x$, répond à la question posée sur $x$

# P vs NP vs PP

- Un problème (de décision) est dans P (polynomial-time), s'il existe un algorithme (déterministe) à temps polynomial, qui, pour toute instance $x$, répond à la question posée sur $x$
- Un problème (de décision) est dans NP (nondeterministic polynomial-time) s'il existe un algorithme nondéterministe à temps polynomial, qui, pour toute instance $x$, "répond" à la question posée sur $x$

# P vs NP vs PP

- Un problème (de décision) est dans P (polynomial-time), s'il existe un algorithme (déterministe) à temps polynomial, qui, pour toute instance $x$, répond à la question posée sur $x$
- Un problème (de décision) est dans NP (nondeterministic polynomial-time) s'il existe un algorithme nondéterministe à temps polynomial, qui, pour toute instance $x$, "répond" à la question posée sur $x$
- Un problème de décision est dans PP (probabilistic polynomial-time) s'il existe un algorithme probabiliste, à temps polynomial, tel que
  - sur toute instance pour laquelle la réponse est `Oui`, la probabilité d'erreur est strictement bornée par $1/2$ ;
  - sur toute instance pour laquelle la réponse est `Non`, la probabilité d'erreur est bornée par $1/2$.

# P vs NP vs PP

- Un problème (de décision) est dans P (polynomial-time), s'il existe un algorithme (déterministe) à temps polynomial, qui, pour toute instance $x$, répond à la question posée sur $x$
- Un problème (de décision) est dans NP (nondeterministic polynomial-time) s'il existe un algorithme nondéterministe à temps polynomial, qui, pour toute instance $x$, "répond" à la question posée sur $x$
- Un problème de décision est dans PP (probabilistic polynomial-time) s'il existe un algorithme probabiliste, à temps polynomial, tel que
  - sur toute instance pour laquelle la réponse est `Oui`, la probabilité d'erreur est strictement bornée par $1/2$ ;
  - sur toute instance pour laquelle la réponse est `Non`, la probabilité d'erreur est bornée par $1/2$.
- **Exercice :** pourquoi le "strictement" est-il important ?

# Quelques exemples

- **Primalité** est dans NP : “Deviner $d'$ ; si $d'$ divise $m$, retourner `Oui`, sinon retourner `Non`”

# Quelques exemples

- **Primalité** est dans NP : “Deviner $d'$ ; si $d'$ divise $m$, retourner `Oui`, sinon retourner `Non`”
- **Primalité** est aussi dans P, mais ce n’est pas facile à voir (Agrawal, Kayal, Saxena 2002)

# Quelques exemples

- **Primalité** est dans NP : “Deviner $d'$ ; si $d'$ divise $m$, retourner `Oui`, sinon retourner `Non`”
- **Primalité** est aussi dans P, mais ce n’est pas facile à voir (Agrawal, Kayal, Saxena 2002)
- **PetitDiviseur** est aussi dans NP (on devine $d' \leq d$ au lieu de $d' < m$), mais non connu pour être dans P (cela permettrait de factoriser $m$ en produit de facteurs premiers, en temps polynomial)

# Premières inclusions

- $P \subset NP$ : un algorithme déterministe est un cas particulier d'algorithme nondéterministe.

# Premières inclusions

- $P \subset NP$ : un algorithme déterministe est un cas particulier d'algorithme nondéterministe.
- (la question de l'inclusion inverse ne semble pas réglée)

# Premières inclusions

- $P \subset NP$ : un algorithme déterministe est un cas particulier d'algorithme nondéterministe.
- (la question de l'inclusion inverse ne semble pas réglée)
- $P \subset PP$ : un algorithme déterministe est un cas particulier d'algorithme probabiliste de probabilité d'erreur 0

# Premières inclusions

- $P \subset NP$ : un algorithme déterministe est un cas particulier d'algorithme nondéterministe.
- (la question de l'inclusion inverse ne semble pas réglée)
- $P \subset PP$ : un algorithme déterministe est un cas particulier d'algorithme probabiliste de probabilité d'erreur 0
- $NP \subset PP$ : à partir d'un algorithme nondéterministe, on obtient un algorithme de probabilité d'erreur strictement inférieure à $1/2$ de la manière suivante :
  - remplacer chaque "Faire X ou faire Y" par "Si `flip()` vaut 1, faire X ; sinon, faire Y"
  - remplacer chaque "Retourner `Non`" par "Si `flip()` vaut 1, retourner `Oui` ; sinon, retourner `Non`".

# Probabilité d'erreur bornée (Monte Carlo)

- Pour $0 < p < 1/2$, un problème (de décision) est dans $BPP(p)$ (bounded-error probabilistic polynomial-time) s'il existe un algorithme probabiliste, à temps polynomial, de probabilité d'erreur bornée par $p$.

# Probabilité d'erreur bornée (Monte Carlo)

- Pour $0 < p < 1/2$, un problème (de décision) est dans $BPP(p)$ (bounded-error probabilistic polynomial-time) s'il existe un algorithme probabiliste, à temps polynomial, de probabilité d'erreur bornée par $p$.
- **Proposition :** Pour tous $0 < p < p' < 1/2$, $BPP(p) = BPP(p')$.

# Probabilité d'erreur bornée (Monte Carlo)

- Pour $0 < p < 1/2$, un problème (de décision) est dans $BPP(p)$ (bounded-error probabilistic polynomial-time) s'il existe un algorithme probabiliste, à temps polynomial, de probabilité d'erreur bornée par $p$.
- **Proposition :** Pour tous $0 < p < p' < 1/2$, $BPP(p) = BPP(p')$.
- **Preuve :** ($BPP(p) \subset BPP(p')$ est trivial) À partir d'un algorithme $A$ de probabilité d'erreur $p'$, on fabrique un algorithme de probabilité d'erreur $p$ en répétant $A$ suffisamment de fois, et en retournant la réponse la plus fréquente. À $p$ et $p'$ fixés, $O(1)$ répétitions suffisent (**Exercice !**)

# Probabilité d'erreur bornée (Monte Carlo)

- Pour $0 < p < 1/2$, un problème (de décision) est dans $BPP(p)$ (bounded-error probabilistic polynomial-time) s'il existe un algorithme probabiliste, à temps polynomial, de probabilité d'erreur bornée par $p$.
- **Proposition :** Pour tous $0 < p < p' < 1/2$, $BPP(p) = BPP(p')$.
- **Preuve :** ($BPP(p) \subset BPP(p')$ est trivial) À partir d'un algorithme $A$ de probabilité d'erreur $p'$, on fabrique un algorithme de probabilité d'erreur $p$ en répétant $A$ suffisamment de fois, et en retournant la réponse la plus fréquente. À $p$ et $p'$ fixés, $O(1)$ répétitions suffisent (**Exercice !**)
- La même chose reste vraie si on autorise $p'$ à être, polynomialement proche de $1/2$, et aussi si on demande que $p$ soit exponentiellement proche de 0 ($p = O(c^n)$, mais le nombre de répétition n'est plus $O(1)$.

# Erreur unilatérale : RP

- Un algorithme probabiliste est **à erreur unilatérale** (de probabilité d'erreur bornée par $p$) s'il ne peut se tromper "que dans un seul sens" (par excès de prudence) :
  - pour toute instance $x$ pour laquelle la réponse est `Oui`, la probabilité d'erreur doit être majorée par $p$ ;
  - pour toute instance $x$ pour laquelle la réponse est `Non`, la probabilité d'erreur doit être 0.

# Erreur unilatérale : RP

- Un algorithme probabiliste est **à erreur unilatérale** (de probabilité d'erreur bornée par $p$) s'il ne peut se tromper "que dans un seul sens" (par excès de prudence) :
  - pour toute instance $x$ pour laquelle la réponse est `Oui`, la probabilité d'erreur doit être majorée par $p$ ;
  - pour toute instance $x$ pour laquelle la réponse est `Non`, la probabilité d'erreur doit être 0.
- Un problème est dans $RP(p)$ (randomized polynomial-time) s'il existe pour ce problème un algorithme à temps polynomial, à erreur unilatérale, de probabilité d'erreur bornée par $p$.

# Erreur unilatérale : RP

- Un algorithme probabiliste est **à erreur unilatérale** (de probabilité d'erreur bornée par $p$) s'il ne peut se tromper "que dans un seul sens" (par excès de prudence) :
  - pour toute instance $x$ pour laquelle la réponse est `Oui`, la probabilité d'erreur doit être majorée par $p$ ;
  - pour toute instance $x$ pour laquelle la réponse est `Non`, la probabilité d'erreur doit être 0.
- Un problème est dans $RP(p)$ (randomized polynomial-time) s'il existe pour ce problème un algorithme à temps polynomial, à erreur unilatérale, de probabilité d'erreur bornée par $p$.
- **Proposition :** pour tous $0 < p < p' < 1$, $RP(p) = RP(p')$.

# Erreur unilatérale : RP

- Un algorithme probabiliste est **à erreur unilatérale** (de probabilité d'erreur bornée par $p$) s'il ne peut se tromper "que dans un seul sens" (par excès de prudence) :
  - pour toute instance $x$ pour laquelle la réponse est `Oui`, la probabilité d'erreur doit être majorée par $p$ ;
  - pour toute instance $x$ pour laquelle la réponse est `Non`, la probabilité d'erreur doit être 0.
- Un problème est dans $RP(p)$ (randomized polynomial-time) s'il existe pour ce problème un algorithme à temps polynomial, à erreur unilatérale, de probabilité d'erreur bornée par $p$.
- **Proposition :** pour tous $0 < p < p' < 1$, $RP(p) = RP(p')$.
- **Preuve :** (c'est encore plus simple que pour BPP)

# Erreur unilatérale : RP

- Un algorithme probabiliste est **à erreur unilatérale** (de probabilité d'erreur bornée par $p$) s'il ne peut se tromper "que dans un seul sens" (par excès de prudence) :
  - pour toute instance $x$ pour laquelle la réponse est `Oui`, la probabilité d'erreur doit être majorée par $p$ ;
  - pour toute instance $x$ pour laquelle la réponse est `Non`, la probabilité d'erreur doit être 0.
- Un problème est dans $RP(p)$ (randomized polynomial-time) s'il existe pour ce problème un algorithme à temps polynomial, à erreur unilatérale, de probabilité d'erreur bornée par $p$.
- **Proposition :** pour tous $0 < p < p' < 1$, $RP(p) = RP(p')$.
- **Preuve :** (c'est encore plus simple que pour BPP)
- Un exemple de problème raisonnablement naturel, candidat à être dans RP mais pas dans P : étant donnés deux polynômes de plusieurs variables, donnés par des expressions arithmétiques (non nécessairement développées), dire s'ils sont distincts.

# Probabilité d'erreur nulle : ZPP (Las Vegas)

- Est-ce qu'on peut demander une probabilité d'erreur *nulle* ?

# Probabilité d'erreur nulle : ZPP (Las Vegas)

- Est-ce qu'on peut demander une probabilité d'erreur *nulle* ?
- Pas vraiment : si on a un algorithme à temps polynomial, et de probabilité d'erreur nulle, alors on a aussi un algorithme *déterministe* à temps polynomial.

# Probabilité d'erreur nulle : ZPP (Las Vegas)

- Est-ce qu'on peut demander une probabilité d'erreur *nulle* ?
- Pas vraiment : si on a un algorithme à temps polynomial, et de probabilité d'erreur nulle, alors on a aussi un algorithme *déterministe* à temps polynomial. **Preuve :** il suffit de remplacer tous les "Si `flip()` vaut 1 faire X, sinon faire Y" par "Faire X" !

# Probabilité d'erreur nulle : ZPP (Las Vegas)

- Est-ce qu'on peut demander une probabilité d'erreur *nulle* ?
- Pas vraiment : si on a un algorithme à temps polynomial, et de probabilité d'erreur nulle, alors on a aussi un algorithme *déterministe* à temps polynomial. **Preuve :** il suffit de remplacer tous les "Si `flip()` vaut 1 faire X, sinon faire Y" par "Faire X" !
- Pour obtenir une classe non trivialement égale à P, on ne contraint que le *temps moyen* : un problème est dans ZPP s'il existe un algorithme probabiliste, à *temps moyen polynomial*, et de probabilité d'erreur 0, pour ce problème.

# Probabilité d'erreur nulle : ZPP (Las Vegas)

- Est-ce qu'on peut demander une probabilité d'erreur *nulle* ?
- Pas vraiment : si on a un algorithme à temps polynomial, et de probabilité d'erreur nulle, alors on a aussi un algorithme *déterministe* à temps polynomial. **Preuve :** il suffit de remplacer tous les "Si `flip()` vaut 1 faire X, sinon faire Y" par "Faire X" !
- Pour obtenir une classe non trivialement égale à P, on ne contraint que le *temps moyen* : un problème est dans ZPP s'il existe un algorithme probabiliste, à *temps moyen polynomial*, et de probabilité d'erreur 0, pour ce problème.
- (Avant 2002, un problème candidat à être dans ZPP sans être dans P était **Primalité**)

# Probabilité d'erreur nulle : ZPP (Las Vegas)

- Est-ce qu'on peut demander une probabilité d'erreur *nulle* ?
- Pas vraiment : si on a un algorithme à temps polynomial, et de probabilité d'erreur nulle, alors on a aussi un algorithme *déterministe* à temps polynomial. **Preuve :** il suffit de remplacer tous les "Si `flip()` vaut 1 faire X, sinon faire Y" par "Faire X" !
- Pour obtenir une classe non trivialement égale à P, on ne contraint que le *temps moyen* : un problème est dans ZPP s'il existe un algorithme probabiliste, à *temps moyen polynomial*, et de probabilité d'erreur 0, pour ce problème.
- (Avant 2002, un problème candidat à être dans ZPP sans être dans P était **Primalité**)
- **Proposition :** $ZPP \subset RP$ (**Indication :** la probabilité qu'une variable aléatoire positive soit supérieure au double de son espérance, est majorée par $1/2$.)

# Complémentation : co-NP, co-RP, co-PP

- Les définitions de NP, RP et PP qu'on a données sont asymétriques : elles distinguent les instances "positives" (pour lesquelles la réponse est `Oui`) des instances "négatives" (pour lesquelles la réponse est `Non`).

# Complémentation : co-NP, co-RP, co-PP

- Les définitions de NP, RP et PP qu'on a données sont asymétriques : elles distinguent les instances "positives" (pour lesquelles la réponse est `Oui`) des instances "négatives" (pour lesquelles la réponse est `Non`).
- Pour un problème de décision $Q$, le problème $\overline{Q}$ est le problème "complémentaire" (qui, pour chaque instance, inverse les réponses `Oui` et `Non`).

# Complémentation : co-NP, co-RP, co-PP

- Les définitions de NP, RP et PP qu'on a données sont asymétriques : elles distinguent les instances "positives" (pour lesquelles la réponse est `Oui`) des instances "négatives" (pour lesquelles la réponse est `Non`).
- Pour un problème de décision $Q$, le problème $\overline{Q}$ est le problème "complémentaire" (qui, pour chaque instance, inverse les réponses `Oui` et `Non`).
- Un problème $Q$ est dans co-NP (resp. co-RP, co-PP) si $\overline{Q}$ est dans NP (resp. RP, PP).

# Complémentation : co-NP, co-RP, co-PP

- Les définitions de NP, RP et PP qu'on a données sont asymétriques : elles distinguent les instances "positives" (pour lesquelles la réponse est `Oui`) des instances "négatives" (pour lesquelles la réponse est `Non`).
- Pour un problème de décision $Q$, le problème $\overline{Q}$ est le problème "complémentaire" (qui, pour chaque instance, inverse les réponses `Oui` et `Non`).
- Un problème $Q$ est dans co-NP (resp. co-RP, co-PP) si $\overline{Q}$ est dans NP (resp. RP, PP).
- NP et co-NP ne sont pas connus pour être égaux, ni RP et co-RP ; en revanche (**Exercice !**), PP=co-PP.

# Complémentation : co-NP, co-RP, co-PP

- Les définitions de NP, RP et PP qu'on a données sont asymétriques : elles distinguent les instances "positives" (pour lesquelles la réponse est `Oui`) des instances "négatives" (pour lesquelles la réponse est `Non`).
- Pour un problème de décision $Q$, le problème $\overline{Q}$ est le problème "complémentaire" (qui, pour chaque instance, inverse les réponses `Oui` et `Non`).
- Un problème $Q$ est dans co-NP (resp. co-RP, co-PP) si $\overline{Q}$ est dans NP (resp. RP, PP).
- NP et co-NP ne sont pas connus pour être égaux, ni RP et co-RP ; en revanche (**Exercice !**), PP=co-PP.
- On a déjà vu que ZPP $\subset$ RP, donc, par complémentation, on a aussi ZPP $\subset$ co-RP ; en fait, on a ZPP $=$ RP $\cap$ co-RP.

# Récapitulatif. . .

- $\mathsf{P} \subset \mathsf{ZPP} \subset \mathsf{RP} \subset \mathsf{NP} \subset \mathit{PP}$

# Récapitulatif...

- $P \subset ZPP \subset RP \subset NP \subset \mathit{PP}$
- Les “bonnes” classes probabilistes sont plutôt ZPP, RP et BPP, à cause des possibilités de réduire arbitrairement la probabilité d'erreur ; mais on manque d'exemples de problèmes “difficiles” dans ces classes (dont l'appartenance à P impliquerait que toute la classe est dans P)

# Récapitulatif...

- $P \subset ZPP \subset RP \subset NP \subset \mathit{PP}$
- Les "bonnes" classes probabilistes sont plutôt ZPP, RP et BPP, à cause des possibilités de réduire arbitrairement la probabilité d'erreur ; mais on manque d'exemples de problèmes "difficiles" dans ces classes (dont l'appartenance à P impliquerait que toute la classe est dans P)
- Est-ce que cela indique que la randomisation (et la possibilité d'accepter une probabilité faible d'erreur) "est inutile" pour résoudre les problèmes ?

# Récapitulatif...

- $P \subset ZPP \subset RP \subset NP \subset \mathit{PP}$
- Les "bonnes" classes probabilistes sont plutôt ZPP, RP et BPP, à cause des possibilités de réduire arbitrairement la probabilité d'erreur ; mais on manque d'exemples de problèmes "difficiles" dans ces classes (dont l'appartenance à P impliquerait que toute la classe est dans P)
- Est-ce que cela indique que la randomisation (et la possibilité d'accepter une probabilité faible d'erreur) "est inutile" pour résoudre les problèmes ?
  - **réponse pessimiste :** oui, ça y ressemble ;

# Récapitulatif...

- $\mathrm{P} \subset \mathrm{ZPP} \subset \mathrm{RP} \subset \mathrm{NP} \subset \mathit{PP}$
- Les "bonnes" classes probabilistes sont plutôt ZPP, RP et BPP, à cause des possibilités de réduire arbitrairement la probabilité d'erreur ; mais on manque d'exemples de problèmes "difficiles" dans ces classes (dont l'appartenance à P impliquerait que toute la classe est dans P)
- Est-ce que cela indique que la randomisation (et la possibilité d'accepter une probabilité faible d'erreur) "est inutile" pour résoudre les problèmes ?
  - **réponse pessimiste :** oui, ça y ressemble ;
  - **réponse pratique :** en basse complexité, il y a des exemples d'algorithmes probabilistes dont la complexité moyenne bat strictement les algorithmes déterministes (**Sélection :** pas d'algorithme déterministe en moins de $3n$ comparaisons, mais existence d'algorithmes randomisés en temps moyen $2n + o(n)$ ; utilisation de randomisation pour économiser les communications)

# Classes de complexité probabilistes(2), simulabilité de lois de probabilités

Philippe Duchon

LaBRI

Aléa 2013

# Résumé de l'épisode précédent

- P : déterministe, polynomial
- RP : polynomial, probabiliste, proba d'erreur unilatérale $\leq p < 1$
- BPP : polynomial, probabiliste, proba d'erreur bilatérale $\leq p < 1/2$
- NP : nondéterministe, polynomial
- PP : polynomial, probabiliste, proba d'erreur (bilatérale) $[<, \leq]1/2$
- inclusions : P $\subset$ RP $\subset$ NP $\subset$ PP, RP $\subset$ BPP

# Probabilité d'erreur nulle : ZPP (Las Vegas)

- Est-ce qu'on peut demander une probabilité d'erreur *nulle* ?

# Probabilité d'erreur nulle : ZPP (Las Vegas)

- Est-ce qu'on peut demander une probabilité d'erreur *nulle* ?
- Pas vraiment : si on a un algorithme à temps polynomial, et de probabilité d'erreur nulle, alors on a aussi un algorithme *déterministe* à temps polynomial.

# Probabilité d'erreur nulle : ZPP (Las Vegas)

- Est-ce qu'on peut demander une probabilité d'erreur *nulle* ?
- Pas vraiment : si on a un algorithme à temps polynomial, et de probabilité d'erreur nulle, alors on a aussi un algorithme *déterministe* à temps polynomial. **Preuve :** il suffit de remplacer tous les "Si `flip()` vaut 1 faire X, sinon faire Y" par "Faire X" !

# Probabilité d'erreur nulle : ZPP (Las Vegas)

- Est-ce qu'on peut demander une probabilité d'erreur *nulle* ?
- Pas vraiment : si on a un algorithme à temps polynomial, et de probabilité d'erreur nulle, alors on a aussi un algorithme *déterministe* à temps polynomial. **Preuve :** il suffit de remplacer tous les "Si `flip()` vaut 1 faire X, sinon faire Y" par "Faire X" !
- Pour obtenir une classe non trivialement égale à P, on ne contraint que le *temps moyen* : un problème est dans ZPP s'il existe un algorithme probabiliste, à *temps moyen polynomial*, et de probabilité d'erreur 0, pour ce problème (algorithme *Las Vegas*).

# Probabilité d'erreur nulle : ZPP (Las Vegas)

- Est-ce qu'on peut demander une probabilité d'erreur *nulle* ?
- Pas vraiment : si on a un algorithme à temps polynomial, et de probabilité d'erreur nulle, alors on a aussi un algorithme *déterministe* à temps polynomial. **Preuve :** il suffit de remplacer tous les "Si `flip()` vaut 1 faire X, sinon faire Y" par "Faire X" !
- Pour obtenir une classe non trivialement égale à P, on ne contraint que le *temps moyen* : un problème est dans ZPP s'il existe un algorithme probabiliste, à *temps moyen polynomial*, et de probabilité d'erreur 0, pour ce problème (algorithme *Las Vegas*).
- (Avant 2002, un problème candidat à être dans ZPP sans être dans P était **Primalité**)

# Probabilité d'erreur nulle : ZPP (Las Vegas)

- Est-ce qu'on peut demander une probabilité d'erreur *nulle* ?
- Pas vraiment : si on a un algorithme à temps polynomial, et de probabilité d'erreur nulle, alors on a aussi un algorithme *déterministe* à temps polynomial. **Preuve :** il suffit de remplacer tous les "Si `flip()` vaut 1 faire X, sinon faire Y" par "Faire X" !
- Pour obtenir une classe non trivialement égale à P, on ne contraint que le *temps moyen* : un problème est dans ZPP s'il existe un algorithme probabiliste, à *temps moyen polynomial*, et de probabilité d'erreur 0, pour ce problème (algorithme *Las Vegas*).
- (Avant 2002, un problème candidat à être dans ZPP sans être dans P était **Primalité**)
- **Proposition :** $ZPP \subset RP$ (**Exercice ! Indication :** la probabilité qu'une variable aléatoire positive soit supérieure au double de son espérance, est majorée par $1/2$.)

# Complémentation : co-NP, co-RP, co-PP

- Les définitions de NP, RP et PP qu'on a données sont asymétriques : elles distinguent les instances "positives" (pour lesquelles la réponse est `Oui`) des instances "négatives" (pour lesquelles la réponse est `Non`).

# Complémentation : co-NP, co-RP, co-PP

- Les définitions de NP, RP et PP qu'on a données sont asymétriques : elles distinguent les instances "positives" (pour lesquelles la réponse est `Oui`) des instances "négatives" (pour lesquelles la réponse est `Non`).
- Pour un problème de décision $Q$, le problème $\overline{Q}$ est le problème "complémentaire" (qui, pour chaque instance, inverse les réponses `Oui` et `Non`).

# Complémentation : co-NP, co-RP, co-PP

- Les définitions de NP, RP et PP qu'on a données sont asymétriques : elles distinguent les instances "positives" (pour lesquelles la réponse est `Oui`) des instances "négatives" (pour lesquelles la réponse est `Non`).
- Pour un problème de décision $Q$, le problème $\overline{Q}$ est le problème "complémentaire" (qui, pour chaque instance, inverse les réponses `Oui` et `Non`).
- Un problème $Q$ est dans co-NP (resp. co-RP, co-PP) si $\overline{Q}$ est dans NP (resp. RP, PP).

# Complémentation : co-NP, co-RP, co-PP

- Les définitions de NP, RP et PP qu'on a données sont asymétriques : elles distinguent les instances "positives" (pour lesquelles la réponse est `Oui`) des instances "négatives" (pour lesquelles la réponse est `Non`).
- Pour un problème de décision $Q$, le problème $\overline{Q}$ est le problème "complémentaire" (qui, pour chaque instance, inverse les réponses `Oui` et `Non`).
- Un problème $Q$ est dans co-NP (resp. co-RP, co-PP) si $\overline{Q}$ est dans NP (resp. RP, PP).
- NP et co-NP ne sont pas connus pour être égaux, ni RP et co-RP ; en revanche (**Exercice !**), PP=co-PP.

# Complémentation : co-NP, co-RP, co-PP

- Les définitions de NP, RP et PP qu'on a données sont asymétriques : elles distinguent les instances "positives" (pour lesquelles la réponse est `Oui`) des instances "négatives" (pour lesquelles la réponse est `Non`).
- Pour un problème de décision $Q$, le problème $\overline{Q}$ est le problème "complémentaire" (qui, pour chaque instance, inverse les réponses `Oui` et `Non`).
- Un problème $Q$ est dans co-NP (resp. co-RP, co-PP) si $\overline{Q}$ est dans NP (resp. RP, PP).
- NP et co-NP ne sont pas connus pour être égaux, ni RP et co-RP ; en revanche (**Exercice !**), PP=co-PP.
- On a déjà vu que ZPP $\subset$ RP, donc, par complémentation, on a aussi ZPP $\subset$ co-RP ; en fait, on a ZPP $=$ RP $\cap$ co-RP.

# Récapitulatif. . .

- $\mathsf{P} \subset \mathsf{ZPP} \subset \mathsf{RP} \subset \mathsf{NP} \subset \mathit{PP}$

# Récapitulatif...

- $P \subset ZPP \subset RP \subset NP \subset \mathit{PP}$
- Les “bonnes” classes probabilistes sont plutôt ZPP, RP et BPP, à cause des possibilités de réduire arbitrairement la probabilité d'erreur (tout en gardant le caractère à peu près polynomial de la complexité) ; mais on manque d'exemples de problèmes “difficiles” dans ces classes (dont l'appartenance à P impliquerait que toute la classe est dans P)

# Récapitulatif...

- $P \subset ZPP \subset RP \subset NP \subset \mathit{PP}$
- Les "bonnes" classes probabilistes sont plutôt ZPP, RP et BPP, à cause des possibilités de réduire arbitrairement la probabilité d'erreur (tout en gardant le caractère à peu près polynomial de la complexité) ; mais on manque d'exemples de problèmes "difficiles" dans ces classes (dont l'appartenance à P impliquerait que toute la classe est dans P)
- Est-ce que cela indique que la randomisation (et la possibilité d'accepter une probabilité faible d'erreur) "est inutile" pour résoudre les problèmes ?

# Récapitulatif...

- $\mathrm{P} \subset \mathrm{ZPP} \subset \mathrm{RP} \subset \mathrm{NP} \subset \mathit{PP}$
- Les “bonnes” classes probabilistes sont plutôt ZPP, RP et BPP, à cause des possibilités de réduire arbitrairement la probabilité d'erreur (tout en gardant le caractère à peu près polynomial de la complexité) ; mais on manque d'exemples de problèmes “difficiles” dans ces classes (dont l'appartenance à P impliquerait que toute la classe est dans P)
- Est-ce que cela indique que la randomisation (et la possibilité d'accepter une probabilité faible d'erreur) “est inutile” pour résoudre les problèmes ?
  - **réponse pessimiste :** oui, ça y ressemble ;

# Récapitulatif...

- $P \subset ZPP \subset RP \subset NP \subset PP$
- Les "bonnes" classes probabilistes sont plutôt ZPP, RP et BPP, à cause des possibilités de réduire arbitrairement la probabilité d'erreur (tout en gardant le caractère à peu près polynomial de la complexité) ; mais on manque d'exemples de problèmes "difficiles" dans ces classes (dont l'appartenance à P impliquerait que toute la classe est dans P)
- Est-ce que cela indique que la randomisation (et la possibilité d'accepter une probabilité faible d'erreur) "est inutile" pour résoudre les problèmes ?
  - **réponse pessimiste :** oui, ça y ressemble ;
  - **réponse pratique :** en basse complexité, il y a des exemples d'algorithmes probabilistes dont la complexité moyenne bat strictement les algorithmes déterministes

# “Simulabilité” de lois de probabilités

- On garde le même modèle de source d’aléa : séquence de bits aléatoires `flip()`

# “Simulabilité” de lois de probabilités

- On garde le même modèle de source d'aléa : séquence de bits aléatoires `flip()`
- On se donne une loi de probabilités (discrète) $\mu$, et on se demande s'il est possible d'avoir un algorithme qui simule (exactement) la loi.

# “Simulabilité” de lois de probabilités

- On garde le même modèle de source d'aléa : séquence de bits aléatoires `flip()`
- On se donne une loi de probabilités (discrète) $\mu$, et on se demande s'il est possible d'avoir un algorithme qui simule (exactement) la loi.
- **Exemple :** on veut simuler la loi de Bernoulli de paramètre $p$ $(0 < p < 1)$

# “Simulabilité” de lois de probabilités

- On garde le même modèle de source d'aléa : séquence de bits aléatoires `flip()`
- On se donne une loi de probabilités (discrète) $\mu$, et on se demande s'il est possible d'avoir un algorithme qui simule (exactement) la loi.
- **Exemple :** on veut simuler la loi de Bernoulli de paramètre $p$ $(0 < p < 1)$
- Inversement, si `flip()` nous donne des Bernoulli de paramètre $p$, peut-on simuler une Bernoulli de paramètre $1/2$ ?

# Temps borné vs temps moyen borné (le retour)

- Avec des Bernoulli de paramètre $1/2$, et en temps borné, on ne peut former des Bernoulli de paramètre $p$ que pour des valeurs $p = a/2^k$ pour $a$ entier (et c'est facile).

# Temps borné vs temps moyen borné (le retour)

- Avec des Bernoulli de paramètre $1/2$, et en temps borné, on ne peut former des Bernoulli de paramètre $p$ que pour des valeurs $p = a/2^k$ pour $a$ entier (et c'est facile).
- Inversement, à partir de Bernoulli de paramètre $p$, et en temps borné, on ne peut pas former de Bernoulli de paramètre $1/2$ si, par exemple, $p$ est transcendant (tous les événements exprimables à partir de $k$ $B_p$ ont une probabilité qui s'exprime comme un polynôme de degré $k$ en $(p, 1-p)$)

# Temps borné vs temps moyen borné (le retour)

- Avec des Bernoulli de paramètre $1/2$, et en temps borné, on ne peut former des Bernoulli de paramètre $p$ que pour des valeurs $p = a/2^k$ pour $a$ entier (et c'est facile).
- Inversement, à partir de Bernoulli de paramètre $p$, et en temps borné, on ne peut pas former de Bernoulli de paramètre $1/2$ si, par exemple, $p$ est transcendant (tous les événements exprimables à partir de $k$ $B_p$ ont une probabilité qui s'exprime comme un polynôme de degré $k$ en $(p, 1-p)$)
- On renonce au temps borné, et on se résigne à n'avoir "que" du temps moyen borné, ou pire, un algorithme qui termine avec probabilité 1.

# Échauffement : rééquilibrer une pièce

Le “truc” de Von Neumann (1951) :

- $a = \texttt{flip()}$, $b = \texttt{flip()}$
- Si $a \neq b$, retourner $a$
- Sinon, recommencer

# Échauffement : rééquilibrer une pièce

Le “truc” de Von Neumann (1951) :

- $a = \texttt{flip()}$, $b = \texttt{flip()}$
- Si $a \neq b$, retourner $a$
- Sinon, recommencer

- Termine en moyenne en $1/2p(1-p)$ répétitions (géométrique)

# Échauffement : rééquilibrer une pièce

Le “truc” de Von Neumann (1951) :

- $a = \texttt{flip()}$, $b = \texttt{flip()}$
- Si $a \neq b$, retourner $a$
- Sinon, recommencer

- Termine en moyenne en $1/2p(1-p)$ répétitions (géométrique)
- Pas besoin de connaître $p$

# Loi simulable

Essayons une définition :

- Une loi de probabilités $\mu$ (sur les entiers) est *simulable* s'il existe un algorithme (à base de flips, et terminant avec probabilité 1) qui retourne une valeur entière distribuée selon $\mu$.

# Loi simulable

Essayons une définition :

- Une loi de probabilités $\mu$ (sur les entiers) est *simulable* s'il existe un algorithme (à base de flips, et terminant avec probabilité 1) qui retourne une valeur entière distribuée selon $\mu$.
- **Remarque :** l'algorithme ne prend pas d'entrée (c'est un algorithme pour chaque loi !)

# Loi simulable

Essayons une définition :

- Une loi de probabilités $\mu$ (sur les entiers) est *simulable* s'il existe un algorithme (à base de flips, et terminant avec probabilité 1) qui retourne une valeur entière distribuée selon $\mu$.
- **Remarque :** l'algorithme ne prend pas d'entrée (c'est un algorithme pour chaque loi !)
- Avec une telle définition, l'ensemble des lois simulables est, au mieux, dénombrable...

# Tirage d'une Bernoulli $p$

Pour obtenir une variable de Bernoulli de paramètre $p$ :

- Tirer $U$, uniforme sur $[0, 1]$
- Si $U < p$, retourner 1 ; sinon, retourner 0

# Tirage d'une Bernoulli $p$

Pour obtenir une variable de Bernoulli de paramètre $p$ :

- Tirer $U$, uniforme sur $[0, 1]$
- Si $U < p$, retourner 1 ; sinon, retourner 0

**Problème :** Pour obtenir une uniforme, il faudrait, au mieux, une suite infinie de bits aléatoires ($U = \sum_{k \geq 1} B_k / 2^k$)

# Tirage d'une Bernoulli $p$

Pour obtenir une variable de Bernoulli de paramètre $p$ :

- Tirer $U$, uniforme sur $[0, 1]$
- Si $U < p$, retourner 1 ; sinon, retourner 0

**Problème :** Pour obtenir une uniforme, il faudrait, au mieux, une suite infinie de bits aléatoires ($U = \sum_{k \geq 1} B_k / 2^k$) **Faux problème :** On n'a pas besoin de tirer une *infinité* de bits, seulement *assez pour pouvoir comparer $U$ à $p$*

## $p = 0.p_1 p_2 p_3 \ldots$

- $k = 0$, $b = 0$
- Tant que $b = p_k$ :
  - $b = \texttt{flip()}$
  - $k = k + 1$
- Retourner $p_k$

- Termine en $k$ flips avec probabilité $1/2^k$

## $p = 0.p_1 p_2 p_3 \ldots$

- $k = 0,\ b = 0$
- Tant que $b = p_k$ :
  - $b = \texttt{flip()}$
  - $k = k + 1$
- Retourner $p_k$

- Termine en $k$ flips avec probabilité $1/2^k$
- La probabilité de retourner 1 en $k$ flips est $p_k/2^k$

# $p = 0.p_1p_2p_3\ldots$

- $k = 0$, $b = 0$
- Tant que $b = p_k$ :
  - $b = \texttt{flip()}$
  - $k = k + 1$
- Retourner $p_k$

- Termine en $k$ flips avec probabilité $1/2^k$
- La probabilité de retourner 1 en $k$ flips est $p_k/2^k$
- En sommant, ça fait pile $p$

# $p = 0.p_1p_2p_3\ldots$

- $k = 0$, $b = 0$
- Tant que $b = p_k$ :
  - $b = \texttt{flip()}$
  - $k = k + 1$
- Retourner $p_k$

- Termine en $k$ flips avec probabilité $1/2^k$
- La probabilité de retourner 1 en $k$ flips est $p_k/2^k$
- En sommant, ça fait pile $p$
- En moyenne, exactement 2 flips

## $p = 0.p_1p_2p_3\ldots$

- $k = 0$, $b = 0$
- Tant que $b = p_k$ :
  - $b =$ `flip()`
  - $k = k + 1$
- Retourner $p_k$

- Termine en $k$ flips avec probabilité $1/2^k$
- La probabilité de retourner 1 en $k$ flips est $p_k/2^k$
- En sommant, ça fait pile $p$
- En moyenne, exactement 2 flips
- **Entourloupe :** ça suppose qu'on ait accès au développement binaire de $p$, ou du moins qu'on soit capable de calculer $p_k$.

## $p = 0.p_1p_2p_3\ldots$

- $k = 0$, $b = 0$
- Tant que $b = p_k$ :
  - $b =$ `flip()`
  - $k = k + 1$
- Retourner $p_k$

- Termine en $k$ flips avec probabilité $1/2^k$
- La probabilité de retourner 1 en $k$ flips est $p_k/2^k$
- En sommant, ça fait pile $p$
- En moyenne, exactement 2 flips
- **Entourloupe :** ça suppose qu'on ait accès au développement binaire de $p$, ou du moins qu'on soit capable de calculer $p_k$.
- Il y a au moins un cas "facile" : si $p = a/b$ est rationnel, le développement binaire de $p$ est facile à calculer (ultimement périodique)

# Notion de “réel calculable”

- **Définition :** un réel $x$ est “calculable” s'il existe un algorithme $A_x$ qui prend en entrée un entier $n > 0$, et retourne un rationnel $r$ tel que $|x - r| \leq 1/n$.

# Notion de “réel calculable”

- **Définition :** un réel $x$ est “calculable” s'il existe un algorithme $A_x$ qui prend en entrée un entier $n > 0$, et retourne un rationnel $r$ tel que $|x - r| \leq 1/n$.
- **Exemple :** Tous les rationnels sont trivialement calculables.

# Notion de “réel calculable”

- **Définition :** un réel $x$ est “calculable” s'il existe un algorithme $A_x$ qui prend en entrée un entier $n > 0$, et retourne un rationnel $r$ tel que $|x - r| \leq 1/n$.
- **Exemple :** Tous les rationnels sont trivialement calculables.
- **Exemple :** $\sqrt{2}$, $e = \sum_{n \geq 0} 1/n!$, $\pi^2 = \sum_{n \geq 1} 6/n^2$ sont calculables.

# Notion de “réel calculable”

- **Définition :** un réel $x$ est “calculable” s'il existe un algorithme $A_x$ qui prend en entrée un entier $n > 0$, et retourne un rationnel $r$ tel que $|x - r| \leq 1/n$.
- **Exemple :** Tous les rationnels sont trivialement calculables.
- **Exemple :** $\sqrt{2}$, $e = \sum_{n \geq 0} 1/n!$, $\pi^2 = \sum_{n \geq 1} 6/n^2$ sont calculables.
- **Proposition :** les réels calculables forment un sous-corps dénombrable de $\mathbb{R}$.

# Notion de “réel calculable”

- **Définition :** un réel $x$ est “calculable” s'il existe un algorithme $A_x$ qui prend en entrée un entier $n > 0$, et retourne un rationnel $r$ tel que $|x - r| \leq 1/n$.
- **Exemple :** Tous les rationnels sont trivialement calculables.
- **Exemple :** $\sqrt{2}$, $e = \sum_{n \geq 0} 1/n!$, $\pi^2 = \sum_{n \geq 1} 6/n^2$ sont calculables.
- **Proposition :** les réels calculables forment un sous-corps dénombrable de $\mathbb{R}$.
- **Remarque :** Il faut se méfier de cette définition, si on s'imagine qu'on peut décrire n'importe quel réel calculable par la donnée d'un algorithme $A_x$ on va au devant de cruelles déconvenues (on ne peut en particulier pas déduire de $A_x$ la réponse à la question “est-ce que $x$ est nul ?”)

# Calculabilité et simulabilité (1)

- **Proposition :** Si un réel $p$, $0 \leq p \leq 1$, est calculable, alors la loi de Bernoulli de paramètre $p$ est simulable.

# Calculabilité et simulabilité (1)

- **Proposition :** Si un réel $p$, $0 \leq p \leq 1$, est calculable, alors la loi de Bernoulli de paramètre $p$ est simulable.
- **Lemme :** Si (et seulement si) $x$ est calculable, alors il existe des algorithmes $L_x$ et $U_x$ qui prennent en entrée un entier $n > 0$, et retournent des valeurs approchées, respectivement par défaut et par excès, à $1/n$ près, de $x$.

# Calculabilité et simulabilité (1)

- **Proposition :** Si un réel $p$, $0 \leq p \leq 1$, est calculable, alors la loi de Bernoulli de paramètre $p$ est simulable.
- **Lemme :** Si (et seulement si) $x$ est calculable, alors il existe des algorithmes $L_x$ et $U_x$ qui prennent en entrée un entier $n > 0$, et retournent des valeurs approchées, respectivement par défaut et par excès, à $1/n$ près, de $x$.
- **Preuve (de la prop.) :** Pour $p$ rationnel, on l'a déjà fait. Pour $p$ irrationnel, les algorithmes $L_p$ et $U_p$ permettent de construire un algorithme qui prend en entrée un entier $n$, et retourne la suite des $n$ premiers bits du développement binaire de $p$ (et on peut utiliser cet algorithme pour mettre en oeuvre le schéma de simulation par développement binaire)

# Calculabilité est simulabilité (2)

Réciproquement :

- **Proposition :** Si la loi de Bernoulli de paramètre $p$ est simulable, alors $p$ est calculable.

# Calculabilité est simulabilité (2)

Réciproquement :

- **Proposition :** Si la loi de Bernoulli de paramètre $p$ est simulable, alors $p$ est calculable.
- **Preuve :** On donne un algorithme $A_p$ qui approche $p$...
  - $k = 1$
  - En *simulant* l'algorithme de génération aléatoire, calculer les probabilités $u_k$, $z_k$, $r_k$ que l'algorithme, respectivement, retourne 1 en au plus $k$ flips, retourne 0 en au plus $k$ flips, utilise plus de $k$ flips (on a $u_k + z_k + r_k = 1$)
  - Si $r_k \leq 1/n$, retourner $u_k$ ; sinon, $k = k + 1$ et recommencer.

  (le fait que l'algorithme de génération aléatoire termine avec probabilité 1, implique qu'il existe un $k$ tel que $r_k \leq 1/n$, donc que notre algorithme $A_p$ termine pour toute valeur de $n$)

# Lois simulables

- On a envie d'avancer qu'une loi $\mu = (\mu_k)_{k \geq 0}$ est simulable, si et seulement si tous les $\mu_k$ sont calculables.

# Lois simulables

- On a envie d'avancer qu'une loi $\mu = (\mu_k)_{k \geq 0}$ est simulable, si et seulement si tous les $\mu_k$ sont calculables.
- C'est certainement une condition nécessaire : un simulateur de la loi $\mu$ fournit, pour chaque $k$, un simulateur de la loi de Bernoulli de paramètre $\mu_k$

# Lois simulables

- On a envie d'avancer qu'une loi $\mu = (\mu_k)_{k\geq 0}$ est simulable, si et seulement si tous les $\mu_k$ sont calculables.
- C'est certainement une condition nécessaire : un simulateur de la loi $\mu$ fournit, pour chaque $k$, un simulateur de la loi de Bernoulli de paramètre $\mu_k$
- *Il me semble* que ce n'est pas réellement suffisant, la "bonne" condition suffisante est que les $mu_k$ soient *tous calculables par un même algorithme* (existence d'un algorithme prenant en entrée $k$ et $n$, et retournant une valeur approchée à $1/n$ près de $\mu_k$).

# Et les lois à densité ?

- Pas évident de définir ce qu'on a envie d'appeler un simulateur (exact ?) de loi à densité (comment est-ce qu'on décrit le résultat ?)

# Et les lois à densité ?

- Pas évident de définir ce qu'on a envie d'appeler un simulateur (exact ?) de loi à densité (comment est-ce qu'on décrit le résultat ?)
- Une idée (présente chez [Knuth-Yao, 1976], mais sans considérations d'algorithmique) : l'algorithme, avec probabilité 1, s'arrête au bout d'un temps fini en retournant un entier $n$ et une suite de bits $b_1, \ldots, b_k$, avec la signification suivante : conditionnellement à $k$,

$$2^k \left( n + \sum_{i=1}^{k} \frac{b_i}{2^i} \right)$$

est distribué comme la partie entière d'une variable aléatoire de loi *mu* multipliée par $2^k$ ;

# Et les lois à densité ?

- Pas évident de définir ce qu'on a envie d'appeler un simulateur (exact ?) de loi à densité (comment est-ce qu'on décrit le résultat ?)
- Une idée (présente chez [Knuth-Yao, 1976], mais sans considérations d'algorithmique) : l'algorithme, avec probabilité 1, s'arrête au bout d'un temps fini en retournant un entier $n$ et une suite de bits $b_1, \ldots, b_k$, avec la signification suivante : conditionnellement à $k$,

$$2^k \left( n + \sum_{i=1}^{k} \frac{b_i}{2^i} \right)$$

  est distribué comme la partie entière d'une variable aléatoire de loi *mu* multipliée par $2^k$ ;
- Avec ce genre de définition, des lois comme l'exponentielle de paramètre 1 sont simulables (toujours [Von Neumann, 1951]), et par un algorithme relativement élémentaire.