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)