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) μ, 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) μ, 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) μ, 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 μ (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 μ.

Loi simulable

Essayons une définition :

- Une loi de probabilités μ (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 μ.
- **Remarque :** l'algorithme ne prend pas d'entrée (c'est un algorithme pour chaque loi !)

Loi simulable

Essayons une définition :

- Une loi de probabilités μ (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 μ.
- **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 μ_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 μ_k sont calculables.
- C'est certainement une condition nécessaire : un simulateur de la loi μ fournit, pour chaque k, un simulateur de la loi de Bernoulli de paramètre μ_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 μ_k sont calculables.
- C'est certainement une condition nécessaire : un simulateur de la loi μ fournit, pour chaque k, un simulateur de la loi de Bernoulli de paramètre μ_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 μ_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.