Génération boltzmannienne exacte avec oracles inexacts

Philippe Duchon

LaBRI

Aléa 2011 - 8 mars 2011

Modèle de Boltzmann

- On a une *famille d'objets discrets* $\mathcal{C}$, munie d'une *taille* $|.|$, telle que, pour tout n,

$$\mathcal{C}_n = \{c \in \mathcal{C} : |c| = n\} \text{ soit fini.}$$

- **Modèle de Boltzmann [D., Flajolet, Louchard, Schaeffer]** : pour un réel $x > 0$ fixé, assigne à chaque objet $c \in \mathcal{C}$ une probabilité *proportionnelle à* $x^{|c|}$ (ou à $x^{|c|}/|c|!$ pour des classes d'objets étiquetés), donc

$$\mathbb{P}_{\mathcal{C},x}(C = c) = \frac{x^{|c|}}{C(x)} \quad \mathbb{P}'_{\mathcal{C},x}(C = c) = \frac{x^{|c|}/|c|!}{\hat{C}(x)}$$

Modèle de Boltzmann

- La *taille de l'objet obtenue* est aléatoire, mais *deux objets de même taille ont la même probabilité d'apparaître*.
- Le choix du paramètre x (entre 0 et le rayon de convergence de la série génératrice) permet au moins de "jouer" sur la taille moyenne.
- La taille moyenne est $\frac{xC'(x)}{C(x)}$, on peut donc chercher si, par exemple, il est possible de trouver x tel qu'elle soit égale à n (et il se trouve que, lorsqu'il existe, ce x maximise la probabilité que la taille soit égale à n).

Générateurs de Boltzmann

Ce qui fait l'attrait du modèle de Boltzmann : *des constructions récursives classiques, se traduisent en équations sur les séries génératrices, et en générateurs de Boltzmann efficaces*, qui se ramènent à la simulation d'un petit nombre de lois discrètes.

- Union disjointe, $\mathcal{C} = \mathcal{A} \oplus \mathcal{B}$: *loi de Bernoulli*

GenC(x)

```
Si Bern(A(x)/(A(x)+B(x)))=1 alors
   Retourner GenA(x)
Sinon
   Retourner GenB(x)
FinSi
```

Générateurs de Boltzmann

Produit, $\mathcal{C} = \mathcal{A} \times \mathcal{B}$: indépendance

Gen(x)

Retourner (GenA(x), GenB(x))

Séquence, $\mathcal{C} = \mathrm{Seq}(\mathcal{A})$: $C(x) = \frac{1}{1-A(x)}$, *loi géométrique*

GenC(x)

N := Geom(A(x))−1
Retourner (GenA(x), ..., GenA(x)) (N appels)

Générateurs de Boltzmann

Ensemble (classes étiquetées) : $\mathcal{C} = \mathrm{Set}(\mathcal{A})$, $C(x) = e^{A(x)}$, *loi de Poisson*

GenC(x)

```
N := Poisson(A(x))
Retourner {GenA(x), ..., GenA(x)}  (N appels)
```

Cycles (classes étiquetées) : $\mathcal{C} = \mathrm{Cycle}(\mathcal{A})$, $C(x) = \log(\frac{1}{1-A(x)})$, *loi "logarithmique"* $p_k(\lambda) = \frac{1}{-\log(1-\lambda)}\frac{\lambda^k}{k}$

GenC(x)

```
N := Loga(A(x))
Retourner (GenA(x), ..., GenA(x))  (N appels)
```

Un cadre confortable

- Définition récursive des classes = composition des générateurs
- Tant qu'on prend le paramètre x strictement plus petit que le rayon de convergence ρ des séries, les générateurs s'arrêtent avec probabilité 1, après un nombre d'appels d'espérance finie
- Possibilité de rejet
- On obtient des objets de taille $\Theta(N)$ en temps moyen $\Theta(N)$

C'était trop beau...

- Seule petite difficulté : il faut connaître les valeurs numériques des séries génératrices (et de certaines fonctions numériques de ces valeurs) au point x
- Résolue via le calcul de l'oracle [Pivoteau, Salvy, Soria] (adaptation de la méthode de Newton aux séries génératrices) convergence rapide vers les valeurs numériques, à x fixé $(x < \rho)$.
- **La question :** Si je fais une petite erreur sur mes constantes, quel est l'impact sur la loi de probabilités des objets obtenus ?

Deux exemples

- Arbres binaires (Catalan) : $\mathcal{T} = 1 \oplus \mathcal{Z} \times \mathcal{T} \times \mathcal{T}$
  - Une seule constante à calculer : la probabilité du "switch" (feuille ou noeud interne)
  - Si imprécision sur la constante : c'est toujours du Boltzmann

Deux exemples

- Arbres binaires (Catalan) : $\mathcal{T} = 1 \oplus \mathcal{Z} \times \mathcal{T} \times \mathcal{T}$
  - Une seule constante à calculer : la probabilité du "switch" (feuille ou noeud interne)
  - Si imprécision sur la constante : c'est toujours du Boltzmann
- Arbres binaires-ternaires :
  $\mathcal{T} = 1 \oplus \mathcal{Z} \times \mathcal{T} \times \mathcal{T} \oplus \mathcal{Z} \times \mathcal{T} \times \mathcal{T} \times \mathcal{T}$
  - Deux constantes à calculer
  - Si imprécision sur les constantes, on n'est plus uniforme à taille fixée

Modèle de génération aléatoire

- On suppose une source parfaite de bits aléatoires indépendants
- On voudrait *éviter les calculs en nombres flottants*, mais *assurer une simulation exacte*
- C'est le modèle des *machines de Buffon* **[Flajolet, Pelletier, Soria]**
- On utilise un "raccourci" pratique :
  - tirer des variables *uniformes* sur $[0, 1]$
  - les comparer entre elles
  - de manière paresseuse (on ne tire que ce dont on a besoin pour les comparaisons)

La loi de Bernoulli

Il faut retourner 1 avec probabilité p, 0 avec probabilité $1-p$.

Bern(p)

```
x := Rand()
Si x<p alors
  Retourner 1
Sinon
  Retourner 0
FinSi
```

La loi géométrique

Il faut retourner k ($k \in \mathbb{N}^*$) avec probabilité $(1-p)^{k-1}p$.

Geom(p)

```
k := 1
x := Rand()
Tant que x>p faire
   k := k+1
   x := Rand()
FinTantQue
Retourner k
```

La loi de Poisson

Il faut retourner k ($k \in \mathbb{N}$) avec probabilité $p_k(\lambda) = e^{-\lambda}\lambda^k/k!$.

- pour éviter la complication de l'exponentielle, on utilise $\tilde{p}_k(\lambda) = (1-\lambda)\lambda^k/k!$ et un rejet
- $\tilde{p}_k(\lambda)$ est la probabilité que l'on ait $U_{k+1} > \lambda > U_1 > U_2 > \cdots > U_k$
- (si la première montée ne coïncide pas avec la première valeur $> \lambda$, "échec" : on recommence)
- si le paramètre λ est trop grand ($> 1/2$), on utilise la propriété d'addition des lois de Poisson ($\mathcal{P}(\lambda) = \mathcal{P}(1/2) * \mathcal{P}(\lambda - 1/2)$)

La loi de Poisson

Poiss(p)

```
k := 0, y:= 1
x := Rand()
Tant que x<p faire
  Si x<y alors
    k := k+1, y := x
  Sinon
    k := 0, y := 1
  x := Rand()
Retourner k
```

La boucle s'arrête au premier `Rand()` qui renvoie un résultat $> p$: donc un nombre géométrique d'appels ; en moyenne, $1/(1-p)$

La loi logarithmique

- $p_k(\lambda) = \frac{1}{\log(1/(1-\lambda))} \frac{\lambda^k}{k}$
- Comme pour la loi de Poisson, on évite le logarithme en prenant $\tilde{p}_k(\lambda) = (1-\lambda)\lambda^k/k$ et en utilisant un rejet.
- $\tilde{p}_k(\lambda)$ est la probabilité d'un événement

$$U_{k+1} > \lambda > U_1 > \{U_2, \ldots, U_k\}$$

RandC(p)

```
x := Rand(); Tant que x>p faire x := Rand()
Retourner x
```

Loga(p)

```
y := RandC(p)
k := 1, x := Rand()
Tant que x<p faire
   Si x<y alors
      k := k+1
   Sinon
      y := RandC(p)
      k := 1
   x := Rand()
Retourner k
```

Et maintenant ?

- On a réussi à simuler toutes nos lois "de base" uniquement avec des uniformes et des comparaisons entre elles et avec le paramètre
- **Remarque cruciale :** si dans le déroulement de l'algorithme de génération on n'a tiré aucune uniforme entre a et b (avec $a < \lambda < b$), alors le générateur, appelé avec n'importe quelle valeur entre a et b, aurait renvoyé la même chose.
- Variante "intervalle de sécurité" des générateurs : en plus d'une valeur entière, on renvoie *la plus grande valeur tirée inférieure au paramètre* et *la plus petite valeur tirée supérieure au paramètre*.
- Compatible avec un tirage *paresseux* (bit à bit) des uniformes !

Bernoulli avec intervalle de sécurité

Bern(p)

```
x := Rand()
Si x<p alors
  Retourner (1,x,1)
Sinon
  Retourner (0,0,x)
FinSi
```

Géométrique avec intervalle de sécurité

Geom(p)

```
k := 1, b:=1
x := Rand()
Tant que x>p faire
   k := k+1, b := min(b,x)
   x := Rand()
FinTantQue
Retourner (k,x,b)
```

Poisson avec intervalle de sécurité

Poiss(p)

```
k := 0, y:= 1, a := 0
x := Rand()
Tant que x<p faire
   Si x<y alors
      k := k+1, y := x
   Sinon
      k := 0, y := 1, a := max(a,x)
   x := Rand()
Retourner (k,a,x)
```

Logarithmique avec intervalle de sécurité

```
(y,a,b) := RandC(p), k := 1, x := Rand()
Tant que x<p faire
   a := max(a,x)
   Si x<y alors
      k := k+1
   Sinon
      (y,c,d) := RandC(p)
      k := 1, a := max(a,c), b := min(b,d)
   x := Rand()
Retourner (k,a,b)
```

Retour vers Boltzmann

- Si on utilise ces générateurs "à intervalles de sécurité" pour former des générateurs de Boltzmann, on récupère, pour *chaque* constante numérique, un intervalle (intersection des intervalles renvoyés par les appels utilisant cette constante)
- les générateurs, avec n'importe quel jeu de constantes appartenant toutes à leurs intervalles respectifs, auraient renvoyé le même objet
- Pour N appels des générateurs élémentaires, la longueur de l'intervalle de sécurité renvoyé est de l'ordre de $1/N$

Boltzmann "exact" avec oracles "raffinables"

- **Hypothèse :** on dispose pour les constantes d'oracles qui fournissent des **encadrements** garantis des valeurs calculées, et qui peuvent être raffinés à la demande
- Alors on peut modifier les générateurs : dès qu'on détecte qu'une variable uniforme "tombe" dans un encadrement, l'algorithme s'interrompt et raffine l'encadrement
- Des générateurs de Boltzmann renvoient alors des objets distribués exactement selon la distribution de Boltzmann
- En moyenne, N appels à un générateur élémentaire exigeront $\log N$ bits : un intervalle d'encadrement de longueur environ $1/N$

Quelques pistes...

- Décrire avec un peu de précision la loi de la longueur d'intervalle de sécurité, pour N appels (*a priori* : loi exponentielle, normalisée par $1/N$)
- On peut se poser la question inverse : si on dispose d'encadrements des constantes, et que le générateur renvoie des intervalles contenant ces encadrements, que peut-on dire de la loi (conditionnée) du résultat ? (non Boltzmann)