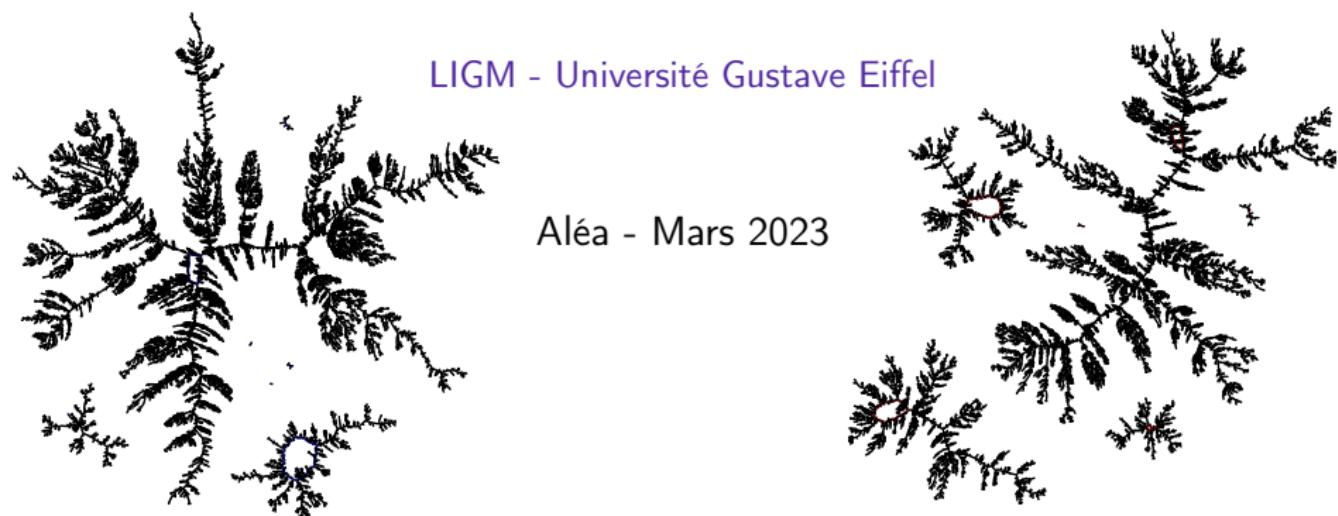


Méthodes automatiques pour la génération aléatoire de structures combinatoires

Carine Pivoteau

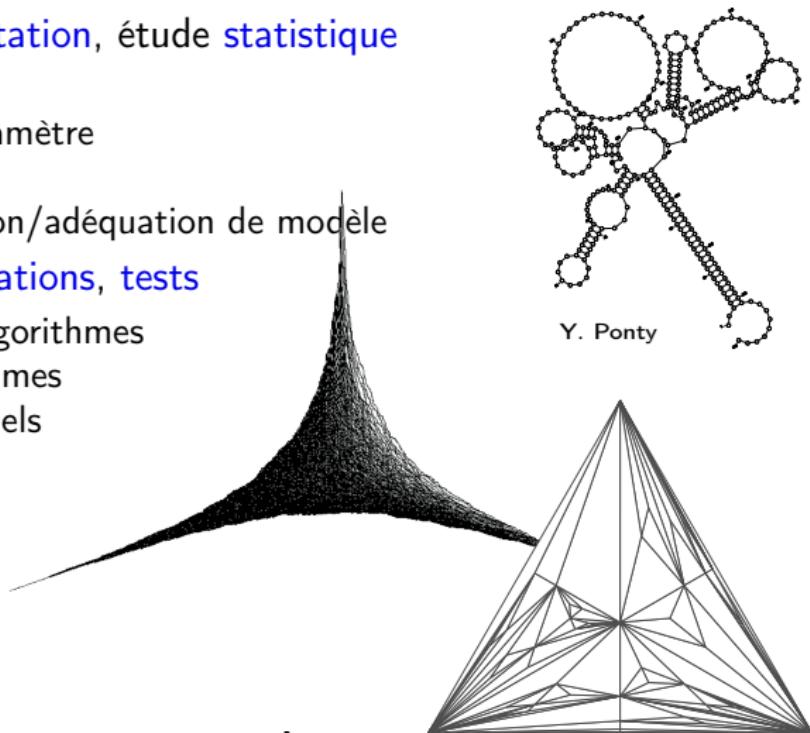
LIGM - Université Gustave Eiffel

Aléa - Mars 2023



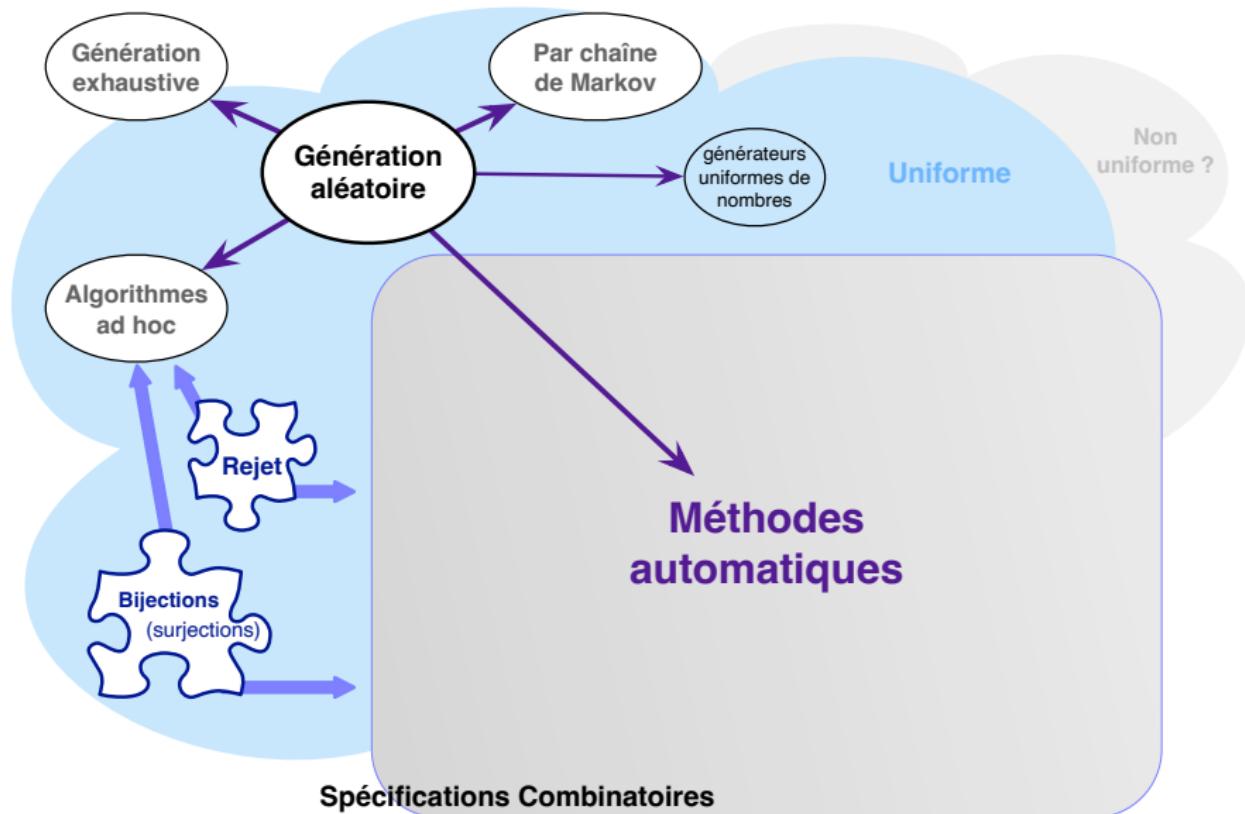
Pourquoi engendrer des structures aléatoires ?

- observation, expérimentation, étude statistique
 - conjectures
 - distributions de paramètre
 - propriétés limites
 - validation : correction/adéquation de modèle
- en informatique : simulations, tests
 - aide à l'analyse d'algorithmes
 - validité des programmes
 - robustesse des logiciels
- autres applications
 - combinatoire
 - bio-informatique
 - physique statistique
 - ...

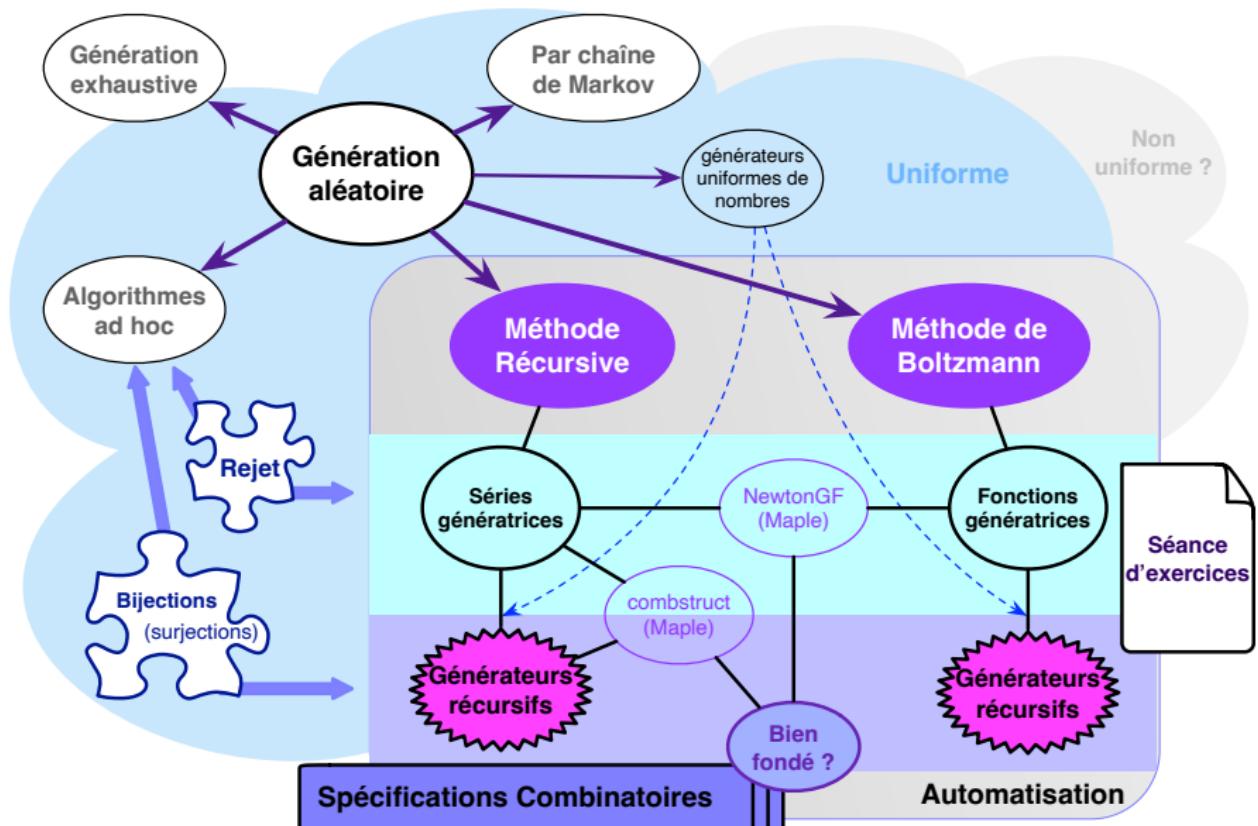


Besoin de générateurs efficaces et automatiques.

Vue d'ensemble



Vue d'ensemble



1 Contexte

2 Méthode Récursive

3 Interlude : créer automatiquement un générateur à partir d'une spécification

4 Méthode de Boltzmann

5 Questions d'implémentation

1 Contexte

- Génération uniforme de structures d'une taille donnée
 - Exemple : engendrer des arbres binaires aléatoires
 - Automatisation : un cadre pour généraliser cette idée

2 Méthode Récursive

3 Interlude : créer automatiquement un générateur à partir d'une spécification

4 Méthode de Boltzmann

5 Questions d'implémentation

Definition (Flajolet-Sedegwick 09)

Une **classe combinatoire** \mathcal{C} est un ensemble *fini* ou *dénombrable* sur lequel est défini une fonction de **taille** telle que :

- la taille d'un élément est positive ou nulle,
- le nombre d'éléments d'une taille donnée est fini.

On appelle les éléments de \mathcal{C} des *structures combinatoires*.

On associe à la classe combinatoire \mathcal{C} une **série génératrice**

$$C(z) = \sum_{n=0}^{\infty} c_n z^n$$

dont le coefficient $c_n = [z^n]C(z)$ énumère les structures de taille n dans la classe \mathcal{C} .

Génération aléatoire uniforme à taille fixée

Génération uniforme dans une classe combinatoire \mathcal{C} **finie** : toute structure γ dans cette classe peut être engendrée avec la probabilité :

$$\mathbb{P}(\gamma) = \frac{1}{|\mathcal{C}|}.$$

Génération uniforme à taille fixée dans \mathcal{C} : pour $n \geq 0$, faire des tirages uniformes dans classe finie \mathcal{C}_n des structures de taille n . Toute structure γ de taille n dans cette classe peut être engendrée avec la probabilité :

$$\mathbb{P}(\gamma) = \frac{1}{c_n}.$$

Génération uniforme pour chaque taille dans \mathcal{C} : la taille n n'est pas fixée, mais pour $n \geq 0$, toutes les structures de taille n ont la même probabilité d'être engendrées.

1 Contexte

- Génération uniforme de structures d'une taille donnée
- Exemple : engendrer des arbres binaires aléatoires
- Automatisation : un cadre pour généraliser cette idée

2 Méthode Récursive

3 Interlude : créer automatiquement un générateur à partir d'une spécification

4 Méthode de Boltzmann

5 Questions d'implémentation

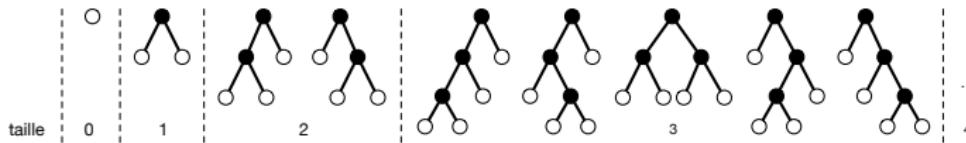
Compter récursivement les arbres binaires

On considère la classe des arbres binaires planaires **dont la taille est le nombre de nœuds internes**. Un arbre binaire de taille n est :

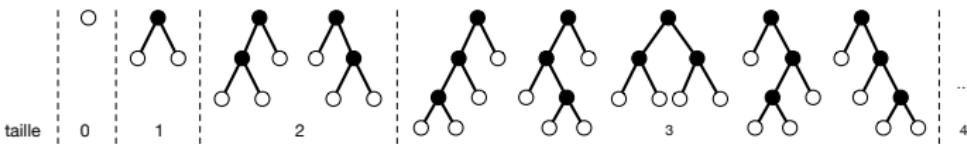
- soit une **feuille** de taille 0, si $n = 0$,
- soit une **racine** de taille 1, qui a 2 fils dont la somme des tailles vaut $n - 1$. Et il y a n possibilités pour ces 2 fils :
 - soit le gauche est de taille 0 et le droit de taille $n - 1$,
 - soit le gauche est de taille 1 et le droit de taille $n - 2$,
 - ...

On en déduit la formule de récurrence pour les nombres de Catalan c_n :

$$c_n = \sum_{i=0}^{n-1} c_i c_{n-i-1}, \quad c_0 = 1$$



Générateur récursif



Pour engendrer un arbre binaire de taille n :

- soit $n = 0$ et on renvoie une feuille (un arbre vide),
- soit $n \geq 1$ et on renvoie une racine ayant un fils gauche γ et un fils droit δ tels que $|\gamma| + |\delta| = n - 1$. On choisit la taille de γ avec la probabilité

$$\mathbb{P}(|\gamma| = i) = \frac{c_i c_{n-i-1}}{c_n}.$$

Ainsi, pour tout arbre T de taille n avec $|\gamma| = k$ et $|\delta| = n - k - 1$, la probabilité d'être engendré est :

$$\mathbb{P}(T) = \frac{c_k c_{n-k-1}}{c_n} \cdot \mathbb{P}(\gamma) \cdot \mathbb{P}(\delta) = \frac{c_k c_{n-k-1}}{c_n} \cdot \frac{1}{c_k} \cdot \frac{1}{c_{n-k-1}} = \frac{1}{c_n}.$$

- En pratique ?
 - Quelle taille d'arbre peut-on atteindre ?
 - Quelle taille d'échantillon (nombre d'arbres) peut-on obtenir ?
 - En combien de temps ?
- [notebook Jupyter ↗](#)
- Et si on veut engendrer...
 - ... des arbres **généraux** planaires ?
 - ... des arbres **non planaires** ?
 - ... des arbres **étiquetés** ?
 - ... d'autres types d'objets ?
- En termes de **complexité** :
 - Est-ce qu'on peut améliorer le pré-calculation ?
 - Et la génération en elle-même ?
 - Et la complexité arithmétique ?
 - Et la complexité en bits ?

Code en python

```
catalan = [1] + [0] * (n-1)
for i in range(1, 1000):
    for j in range(i):
        catalan[i] += catalan[j] * catalan[i-j-1]

def uniform_cut(n) : #choisir i avec probabilité  $p_i = c_i c_{n-i-1} / c_n$ 
    u = random()
    i, s = 0, catalan[0] * catalan[n-1] / catalan[n]
    while u > s :
        i += 1
        s += catalan[i] * catalan[n-i-1] / catalan[n]
    return i

def gen(n):
    if n == 0:
        return []
    i = uniform_cut(n) #  $O(n)$ 
    return ['z', gen(i), gen(n-i-1)]
```

Complexité dans le pire cas : $O(n^2)$ + pré-calcul (ici, $O(n^2)$ également)

1 Contexte

- Génération uniforme de structures d'une taille donnée
- Exemple : engendrer des arbres binaires aléatoires
- Automatisation : un cadre pour généraliser cette idée

2 Méthode Récursive

3 Interlude : créer automatiquement un générateur à partir d'une spécification

4 Méthode de Boltzmann

5 Questions d'implémentation

- spécification combinatoire (grammaire récursive) :

$$\mathcal{Y} = \mathcal{H}(\mathcal{Z}, \mathcal{Y}) \quad \equiv \quad \begin{cases} \mathcal{Y}_1 = \mathcal{H}_1(\mathcal{Z}, \mathcal{Y}_1, \mathcal{Y}_2, \dots, \mathcal{Y}_m), \\ \mathcal{Y}_2 = \mathcal{H}_2(\mathcal{Z}, \mathcal{Y}_1, \mathcal{Y}_2, \dots, \mathcal{Y}_m), \\ \vdots \\ \mathcal{Y}_m = \mathcal{H}_m(\mathcal{Z}, \mathcal{Y}_1, \mathcal{Y}_2, \dots, \mathcal{Y}_m), \end{cases}$$

- constructions combinatoires usuelles (étiquetées ou non) :
 \mathcal{E} , \mathcal{Z} , $+$, \times , séquence, cycle, ensemble (+ contraintes de cardinalité)
- série génératrice automatique pour toute classe combinatoire \mathcal{C} :

$$\text{ordinaire : } C(z) = \sum_{n \geq 0} c_n z^n$$

$$\text{exponentielle : } \hat{C}(z) = \sum_{n \geq 0} C_n \frac{z^n}{n!}$$

où c_n (resp. C_n) est le nombre de structures de \mathcal{C} de taille n (nombre d'atomes). Dans le cas étiqueté, on note c_n le rationnel $c_n = \frac{C_n}{n!}$.

Dictionnaire des séries génératrices

	construction	s.g. ordinaire (non étiquetée)	s.g. exponentielle (étiquetée)
\mathcal{E} ou atome	1 ou \mathcal{Z}	1 ou z	1 ou z
Union	$\mathcal{A} + \mathcal{B}$	$A(z) + B(z)$	$A(z) + B(z)$
Produit	$\mathcal{A} \times \mathcal{B}$	$A(z) \times B(z)$	$A(z) \times B(z)$
Séquence	$\text{Seq}(\mathcal{A})$	$\frac{1}{1 - A(z)}$	$\frac{1}{1 - A(z)}$
Ensemble	$\text{Set/PSet}(\mathcal{A})$	$\exp\left(\sum_{k=1}^{\infty} \frac{(-1)^{k-1}}{k} A(z^k)\right)$	$\exp(A(z))$
Multi-ensemble	$\text{MSet}(\mathcal{A})$	$\exp\left(\sum_{k=1}^{\infty} \frac{1}{k} A(z^k)\right)$	-
Cycle	$\text{Cyc}(\mathcal{A})$	$\sum_{k=1}^{\infty} \frac{\varphi(k)}{k} \log \frac{1}{1 - A(z^k)}$	$\log \frac{1}{1 - A(z)}$

Exemples de spécifications combinatoires

non étiqueté		
$\mathcal{P} = \text{MSet}(\text{Seq}_{\geq 1}(\mathcal{Z}))$	partitions d'entier	$P(z) = \exp \left(\sum_{k=1}^{\infty} \frac{z^k}{k(1-z^k)} \right)$
$\mathcal{Q} = \text{PSet}(\text{Seq}_{\geq 1}(\mathcal{Z}))$	partitions en sommants \neq	$Q(z) = \exp \left(\sum_{k=1}^{\infty} \frac{(-1)^{k-1} z^k}{k(1-z^k)} \right)$
$\mathcal{C} = \text{Seq}(\text{Seq}_{\geq 1}(\mathcal{Z}))$	compositions d'entier	$C(z) = \frac{1}{1 - \frac{z}{1-z}}$
$\mathcal{B} = \mathcal{Z} + \mathcal{B} \times \mathcal{B}$	arbres binaires planaires	$B(z) = z + B(z)^2$
$\mathcal{A} = \mathcal{Z} \times \text{Seq}(\mathcal{A})$	arbres généraux planaires	$A(z) = \frac{z}{1 - A(z)}$
étiqueté		
$\mathcal{P}_e = \text{Set}(\text{Cyc}(\mathcal{Z}))$	permutations	$P_e(z) = e^{\log 1/(1-z)} = \frac{1}{1-z}$
$\begin{cases} \mathcal{S} = \text{Seq}_{\geq 2}(\mathcal{P} + \mathcal{Z}) \\ \mathcal{P} = \text{Set}_{\geq 2}(\mathcal{S} + \mathcal{Z}) \end{cases}$	graphes série-parallèle	$\begin{cases} S(z) = \frac{(P(z)+z)^2}{1-(P(z+z))} \\ P(z) = e^{S(z)+z} - S(z) - z \end{cases}$
$\mathcal{T} = \mathcal{Z} \times \text{Set}(\mathcal{T})$	arbres non planaires	$T(z) = ze^{T(z)}$
$\begin{cases} \mathcal{G} = \text{Set}(\text{Cyc}(\mathcal{T})) \\ \mathcal{T} = \mathcal{Z} \times \text{Set}(\mathcal{T}) \end{cases}$	graphes fonctionnels	$\begin{cases} G(z) = e^{\log \frac{1}{1-T(z)}} = \frac{1}{1-T(z)} \\ T(z) = ze^{T(z)} \end{cases}$

- 1 Contexte
- 2 Méthode Récursive
- 3 Interlude : créer automatiquement un générateur à partir d'une spécification
- 4 Méthode de Boltzmann
- 5 Questions d'implémentation

Idée : *Nijenhuis et Wilf*, 1978

Systématisation : *Flajolet, Zimmermann et Van Cutsem*, 1994

Principe

Pour engendrer aléatoirement une structure de taille n dans la classe \mathcal{C} définie comme “composition” des classes $\mathcal{C}_1, \dots, \mathcal{C}_k$:

- on choisit la taille (et éventuellement, le nombre) des structures de $\mathcal{C}_1, \dots, \mathcal{C}_k$ qui la composent,
- de telle sorte que la somme de leurs tailles soit n ,
- de telle façon que toutes les structures de \mathcal{C} aient une probabilité $\frac{1}{|\mathcal{C}|}$ d'être tirées.

La décomposition récursive de \mathcal{C} fournit à la fois l'algorithme de **dénombrement** et l'algorithme de **génération**.

Les arbres généraux planaires $\mathcal{T} = \mathcal{Z} \times \text{Seq}(\mathcal{T})$

En utilisant la définition récursive d'une séquence, cette spécification devient :

$$\mathcal{T} = \mathcal{Z} \times \mathcal{F} \quad \text{et} \quad \mathcal{F} = \mathcal{E} + \mathcal{T} \times \mathcal{F},$$

où \mathcal{F} représente la classe des forêts. Les coefficients t_n et f_n comptent le nombre d'arbres et de forêts de taille n :

$$t_n = f_n - 1 \quad \text{et} \quad f_n = \begin{cases} 1 & \text{si } n = 0 \\ \sum_{k=1}^n t_k f_{n-k} & \text{sinon.} \end{cases}$$

Générateur :

GenR(\mathcal{T}, n)

renvoyer $\mathcal{Z} \times \text{GenR}(\mathcal{F}, n - 1)$

GenR(\mathcal{F}, n)

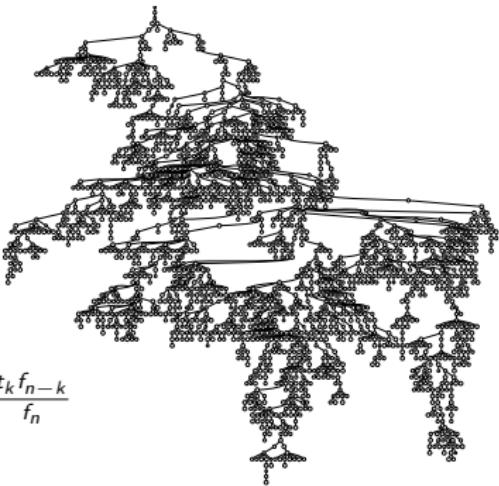
si $n = 0$ alors

renvoyer \mathcal{E}

sinon

choisir s dans $[1, n]$ tel que $\mathbb{P}(s = k) = \frac{t_k f_{n-k}}{f_n}$

renvoyer $\text{GenR}(\mathcal{T}, s) \times \text{GenR}(\mathcal{F}, n - s)$



1 Contexte

2 Méthode Réursive

- Standardiser la spécification
- Compter récursivement
- Engendrer récursivement
- En pratique
- Digression autour des générateurs de chemins

3 Interlude : créer automatiquement un générateur à partir d'une spécification

4 Méthode de Boltzmann

5 Questions d'implémentation

Pré-traitement : mettre la spécification sous forme standard. Même principe que la forme normale de Chomsky pour les grammaires hors contexte.

On réécrit toutes les constructions étiquetées `on` ou `non`, sauf `PSet`, en termes d'**unions** et de **produits binaires**.

Definition (FlZiVC94)

Soit $\mathcal{C} = (\mathcal{C}_1, \dots, \mathcal{C}_m)$ un m -uplet de classes combinatoires. Une *spécification standard* de \mathcal{C} est un système à m équations dont la i ème s'écrit sous la forme :

$$\begin{aligned}\mathcal{C}_i &= \mathcal{E} & \mathcal{C}_i &= \mathcal{Z} & \mathcal{C}_i &= \mathcal{U}_j + \mathcal{U}_k & \mathcal{C}_i &= \mathcal{U}_j \times \mathcal{U}_k \\ \mathcal{C}_i &= \Theta \mathcal{U}_j & \mathcal{C}_i &= \Delta_{\{u(k)\}} \mathcal{U}_j & \Theta \mathcal{C}_i &= \mathcal{U}_j,\end{aligned}$$

où chaque \mathcal{U}_i est dans $\{\mathcal{C}_1, \dots, \mathcal{C}_m\}$ et $u(k)$, pour $k \geq 1$, est une suite d'entiers positifs.

L'opérateur de pointage (étiqueté ou non)

Le symbole Θ désigne l'opérateur de **pointage** (également noté \mathcal{A}^\bullet) :

$$\Theta\mathcal{A} = \bigcup_{n=1}^{\infty} (\mathcal{A}_n \times \{1, 2, \dots, n\}),$$

Une structure de $\Theta\mathcal{A}$ est donc une structure de \mathcal{A} avec un atome pointé.

▷ Correspond à une **dérivation** sur les séries génératrices :

$$\mathcal{C} = \Theta\mathcal{A} \quad \Rightarrow \quad C(z) = \Theta A(z), \quad \text{où } \Theta f(z) = z \cdot \frac{d}{dz} f(z).$$

Par exemple, pour l'atome, l'union et le produit :

$$\mathcal{C} = \mathcal{Z} \quad \Rightarrow \quad \Theta\mathcal{C} = \mathcal{Z},$$

$$\mathcal{C} = \mathcal{A} + \mathcal{B} \quad \Rightarrow \quad \Theta\mathcal{C} = \Theta\mathcal{A} + \Theta\mathcal{B},$$

$$\mathcal{C} = \mathcal{A} \times \mathcal{B} \quad \Rightarrow \quad \Theta\mathcal{C} = \Theta\mathcal{A} \times \mathcal{B} + \mathcal{A} \times \Theta\mathcal{B}.$$

La diagonale (non étiquetée)

La **diagonale** à k éléments d'une classe \mathcal{A} , notée $\Delta^{(k)}\mathcal{A}$, est l'ensemble des k -uplets de structures de \mathcal{A} identiques.
Sa série génératrice est $A(z^k)$.

Étant donnée une suite d'entiers positifs $u(k)$, avec $k \geq 1$, la **diagonale généralisée** est définie par

$$\Delta_{\{u(k)\}} = \sum_{k=1}^{\infty} u(k) \Delta^{(k)},$$

et la série génératrice de $\Delta_{\{u(k)\}}\mathcal{A}$ est $\sum_{k=1}^{\infty} u(k) A(z^k)$.

Pour les ensembles et les cycles, on a les identités combinatoires suivantes :

$$\mathcal{C} = \text{Cyc}(\mathcal{A}) \Rightarrow \Theta\mathcal{C} = \Delta_{\{\varphi(k)\}} \Theta\mathcal{A} \times \text{Seq}(\mathcal{A})$$

$$\mathcal{C} = \text{MSet}(\mathcal{A}) \Rightarrow \Theta\mathcal{C} = \mathcal{C} \times \Delta_{\{1\}} \Theta\mathcal{A}$$

Théorème (FlZiVC94+97)

Toute famille de classes combinatoires décomposables étiquetées et non étiquetées sans PSet admet une spécification standard.

constructions étiquetées	
$\mathcal{C} = \text{Seq}(\mathcal{A})$	$\Rightarrow \mathcal{C} = \mathcal{E} + \mathcal{A} \times \mathcal{C}$
$\mathcal{C} = \text{Cyc}(\mathcal{A})$	$\Rightarrow \Theta\mathcal{C} = \text{Seq}(\mathcal{A}) \times \Theta\mathcal{A}$
$\mathcal{C} = \text{Set}(\mathcal{A})$	$\Rightarrow \Theta\mathcal{C} = \mathcal{C} \times \Theta\mathcal{A}$
constructions non étiquetées	
$\mathcal{C} = \text{Cyc}(\mathcal{A})$	$\Rightarrow \Theta\mathcal{C} = \sum_{k \geq 1} \varphi(k) \Delta^{(k)} \Theta\mathcal{A} \times \text{Seq}(\Delta^{(k)} \mathcal{A})$
$\mathcal{C} = \text{MSet}(\mathcal{A})$	$\Rightarrow \Theta\mathcal{C} = \mathcal{C} \times \sum_{k \geq 1} \Delta^{(k)} \Theta\mathcal{A}$

Il existe des variantes pour chaque contrainte de cardinalité.

Pour PSet, on engendre des MSet et on utilise du **rejet**.

Exemple : graphes fonctionnels

La classe \mathcal{G} des graphes fonctionnels est spécifiée par la grammaire étiquetée

$$\begin{aligned}\mathcal{G} &= \text{Set}(\text{Cyc}(\mathcal{T})) \\ \mathcal{T} &= \mathcal{Z} \times \text{Set}(\mathcal{T}).\end{aligned}$$

La forme standard de cette grammaire est :

$$\Theta\mathcal{G} = \mathcal{G} \times \Theta\mathcal{C}$$

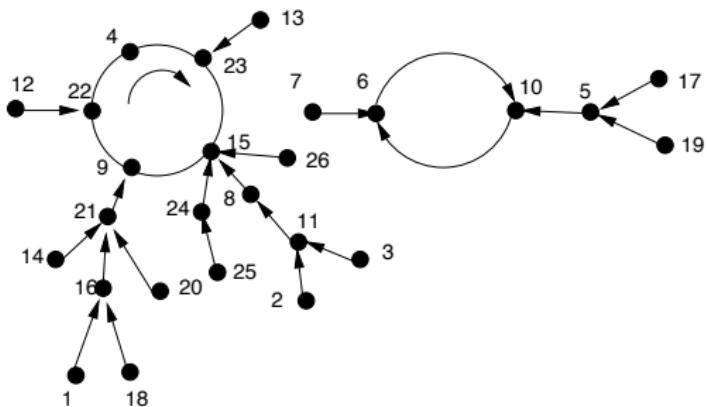
$$\Theta\mathcal{C} = \mathcal{S} \times \Theta\mathcal{T}$$

$$\mathcal{S} = \mathcal{E} + \mathcal{P}$$

$$\mathcal{P} = \mathcal{T} \times \mathcal{S}$$

$$\mathcal{T} = \mathcal{Z} \times \mathcal{F}$$

$$\Theta\mathcal{F} = \mathcal{F} \times \Theta\mathcal{T}.$$



1 Contexte

2 Méthode Réursive

- Standardiser la spécification
- Compter récursivement
- Engendrer récursivement
- En pratique
- Digression autour des générateurs de chemins

3 Interlude : créer automatiquement un générateur à partir d'une spécification

4 Méthode de Boltzmann

5 Questions d'implémentation

Compter les structures étiquetées (ou non)

En étiqueté, on calcule les coefficients normalisés (rationnels).

$$\mathcal{C} = \mathcal{E} \implies c_0 = 1 \text{ et } \forall n \geq 1, c_n = 0$$

$$\mathcal{C} = \mathcal{Z} \implies c_1 = 1 \text{ et } \forall n \neq 1, c_n = 0$$

$$\mathcal{C} = \mathcal{A} + \mathcal{B} \implies c_n = a_n + b_n, \quad \forall n \geq 0$$

$$\mathcal{C} = \mathcal{A} \times \mathcal{B} \implies c_n = \sum_{k=0}^n f_k t_{n-k}, \quad \forall n \geq 0$$

$$\Theta \mathcal{C} = \mathcal{A} \implies c_0 = 0 \text{ et } \forall n \geq 1, c_n = a_n/n$$

$$\mathcal{A} = \Delta^{(k)} \mathcal{B} \implies a_n = b_{n/k} \text{ si } k|n, \quad 0 \text{ sinon.}$$

Pour les ensembles et les cycles, une structure non étiquetée de taille n ne correspond pas forcément à $n!$ structures étiquetées.

▷ Le pointage permet de “retirer” les symétries.

Calculer les n premiers coefficients : $O(n^2)$ opérations arithmétiques, à cause de la somme dans le calcul du produit.

Pour calculer les coefficients (étiquetés ou non) plus efficacement :

- Relax, but don't be too lazy, J. van der Hoeven, 2002 : $O(n \log^2 n)$
- Algorithms for combinatorial structures : Well-founded systems and Newton iterations, C. Pivoteau, B. Salvy, M. Soria, 2012 : $O(n \log n)$

La complexité en bits est en $O(n^{3+\varepsilon})$ car les coefficients (étiquetés) sont de taille $O(n \log n)$. On peut revenir à $O(n^{2+\varepsilon})$ en utilisant des flottants.

- Uniform random generation of decomposable structures using floating-point arithmetic, A. Denise, P. Zimmerman, 1999

1 Contexte

2 Méthode Récursive

- Standardiser la spécification
- Compter récursivement
- Engendrer récursivement
 - En pratique
 - Digression autour des générateurs de chemins

3 Interlude : créer automatiquement un générateur à partir d'une spécification

4 Méthode de Boltzmann

5 Questions d'implémentation

Les générateurs

GenR(\mathcal{C}, n)

cas $\mathcal{C} = \mathcal{E}$: renvoyer \mathcal{E} # uniquement quand $n = 0$

cas $\mathcal{C} = \mathcal{Z}$: renvoyer \mathcal{Z} # uniquement quand $n = 1$

cas $\mathcal{C} = \mathcal{A} + \mathcal{B}$:

$U \leftarrow \text{random}()$

si $U < (a_n/c_n)$ alors

 renvoyer **GenR**(\mathcal{A}, n)

sinon renvoyer **GenR**(\mathcal{B}, n)

cas $\mathcal{C} = \mathcal{A} \times \mathcal{B}$:

$U \leftarrow \text{random}()$

$k \leftarrow 0$

$s \leftarrow (a_0 b_n)/c_n$

tant que $s < U$ faire

$k \leftarrow k + 1$

$s \leftarrow s + (a_k b_{n-k})/c_n$

renvoyer **GenR**(\mathcal{A}, k) \times **GenR**($\mathcal{B}, n - k$)

Les générateurs - suite

GenR(\mathcal{C}, n)

cas $\mathcal{C} = \Theta\mathcal{A}$ ou \mathcal{C} implicitement définie par $\Theta\mathcal{C} = \mathcal{A}$:

renvoyer **GenR**(\mathcal{A}, n)

En pratique, le pointage sert uniquement au calcul des probabilités.

cas $\mathcal{C} = \Delta^{(k)}\mathcal{A}$: # uniquement dans le cas où k divise n

$a = \text{GenR}(\mathcal{A}, n/k)$

renvoyer un k -uplet **de** a

cas $\mathcal{C} = \sum_{k \geq 1} f(k)\Delta^{(k)}\mathcal{A}$:

$U \leftarrow \text{random}()$

$k \leftarrow 1$

$s \leftarrow f(1)a_n/c_n$

tant que $s < U$ **faire**

$k \leftarrow$ prochain diviseur de n

$s \leftarrow s + f(k)a_{n/k}/c_n$

$a = \text{GenR}(\mathcal{A}, n/k)$

renvoyer un k -uplet **de** a

Étiquetage éventuel : on applique une permutation aléatoire sur les atomes.

Complexité de la génération

Complexité (FlZiVC94)

Avec cet algorithme, la **complexité** en nombre d'*opérations arithmétiques* pour engendrer une structure de taille n est en $O(n^2)$ en pire cas.

- L'arbre (binaire) de décomposition d'une structure de taille n a lui-même une taille proportionnelle* à n .
- Chaque nœud de cet arbre peut nécessiter un nombre d'opérations linéaire en la taille de son sous-arbre dans le pire des cas (toujours à cause du produit).
- La complexité est donc bornée par la longueur de cheminement qui est en $O(n^2)$.

(*) En particulier, on contrôle le nombre d' \mathcal{E} dans l'arbre de décomposition.

Amélioration de la complexité pire cas

Boustrophédon : pour engendrer un produit de taille n , on cherche la taille k de la première structure en commençant par $k = 0$, puis $k = n$, puis $k = 1$, puis $k = n - 1$, ...

[notebook Jupyter](#) ↗

On s'arrête donc lorsque l'on a trouvé la taille de la plus petite des deux composantes du produit, en au plus $2 \min(k, n - k) + 2$ étapes.

Complexité (FlZiVC94)

Avec une recherche de type *boustrophédon*, la **complexité** en nombre d'opérations arithmétiques pour engendrer une structure de taille n est en **$O(n \log n)$** en pire cas.

La complexité $f(n)$ pour engendrer un produit de taille n est donc de la forme :

$$f(n) = \max_k(f(k) + f(n - k) + 2 \min(k, n - k)).$$

La complexité en bits est en $O(n^{2+\varepsilon})$ et on peut revenir à $O(n^{1+\varepsilon})$ en utilisant des générateurs en flottant (cf. article d'A. Denise et P. Zimmermann).

1 Contexte

2 Méthode Réursive

- Standardiser la spécification
- Compter récursivement
- Engendrer récursivement
- En pratique
- Digression autour des générateurs de chemins

3 Interlude : créer automatiquement un générateur à partir d'une spécification

4 Méthode de Boltzmann

5 Questions d'implémentation

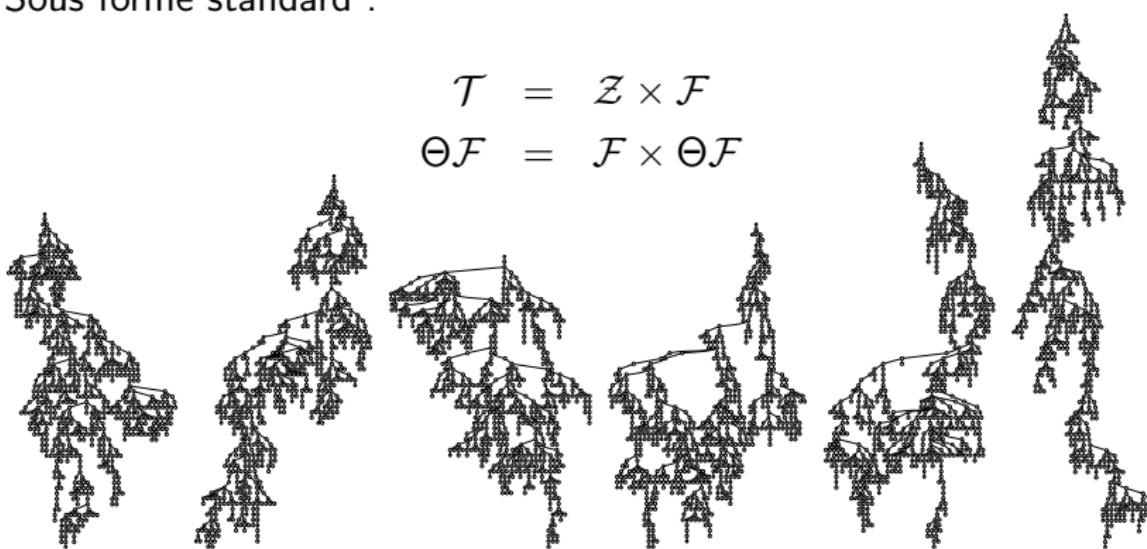
Démo

Générateur aléatoire récursif de arbre généraux non planaires étiquetés :

$$\mathcal{T} = \mathcal{Z} + \mathcal{Z} \times \text{Set}(\mathcal{M}).$$

Sous forme standard :

$$\begin{aligned}\mathcal{T} &= \mathcal{Z} \times \mathcal{F} \\ \Theta\mathcal{F} &= \mathcal{F} \times \Theta\mathcal{F}\end{aligned}$$



Démo

Programmer un générateur aléatoire récursif de **mobiles** étiquetés :

$$\mathcal{M} = \mathcal{Z} + \mathcal{Z} \times \text{Cyc}(\mathcal{M}).$$

Sous forme standard :

$$\mathcal{M} = \mathcal{Z} + \mathcal{Z} \times \mathcal{C}$$

$$\Theta\mathcal{C} = \mathcal{S} \times \Theta\mathcal{M}$$

$$\Theta\mathcal{S} = \mathcal{E} + \mathcal{M} \times \mathcal{S}$$



1 Contexte

2 Méthode Réursive

- Standardiser la spécification
- Compter récursivement
- Engendrer récursivement
- En pratique
- Digression autour des générateurs de chemins

3 Interlude : créer automatiquement un générateur à partir d'une spécification

4 Méthode de Boltzmann

5 Questions d'implémentation

Chemins positifs

$\mathcal{P}_{i,j}$: chemins composés de i pas $(+1, +1)$ et j pas $(-1, +1)$, débutant à l'origine $(0, 0)$ et ne passant pas sous l'axe des abscisses.

$$\mathcal{P}_{0,0} = \mathcal{E}$$

$$\mathcal{P}_{i,j} = \mathcal{P}_{i-1,j} \times \nearrow + \mathcal{P}_{i,j-1} \times \searrow$$

$$\mathcal{P}_{i,j} = \emptyset \text{ si } j > i$$

$$p_{i,j} = \binom{i+j+1}{i+1} \frac{i-j+1}{i+j+1}.$$

Gen(\mathcal{P}, i, j)

si $i = j = 0$ alors

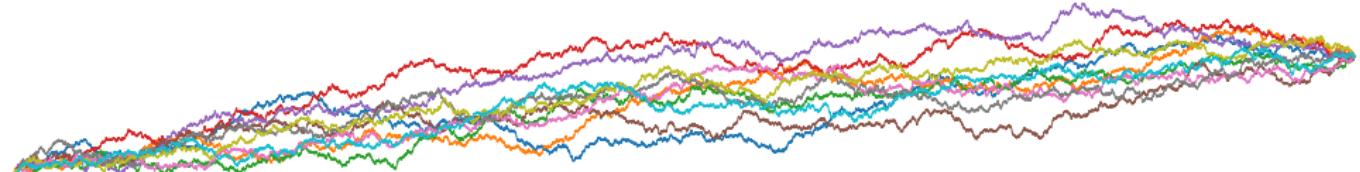
renvoyer \mathcal{E}

sinon

$U \leftarrow \text{random}()$

si $U < \frac{i-j}{i-j+1} \frac{i+1}{i+j}$ alors renvoyer **Gen**($\mathcal{P}, i-1, j$) suivi de \nearrow

sinon renvoyer **Gen**($\mathcal{P}, i, j-1$) suivi de \searrow



Chemin de Motzkin et facteurs gauches

Chemin de Motzkin : chemin positif composé de pas $(+1, +1)$, $(-1, +1)$ et $(0, +1)$, qui commence en $(0, 0)$ et se termine sur l'axe des abscisses.

Facteur gauche de chemin de Motzkin : n'importe quel chemin composé de ces trois types de pas pouvant se compléter en un chemin de Motzkin.



La classe \mathcal{F} des facteurs gauches de chemins de Motzkin est définie par la spécification :

$$\mathcal{M} = \mathcal{E} + \rightarrow \mathcal{M} + \nearrow \mathcal{M} \searrow \mathcal{M}, \quad \text{et} \quad \mathcal{F} = \mathcal{M} + \mathcal{M} \nearrow \mathcal{F}.$$

Génération récursive d'un facteur gauche de longueur n : $O(n \log n)$

Engendrer des facteurs gauches de mots de Motzkin = étape préalable pour engendrer des animaux dirigés (*E. Barcucci, R. Pinzani, R. Sprugnoli, 1994*).

Rejet, rejet anticipé

Génération par **rejet** :

- faire des tirages dans un sur-ensemble de la classe \mathcal{F} visée et rejeter les objets qui ne sont pas dans \mathcal{F} .
- Complexité : on ajoute le coût de la vérification et la génération de tous les objets engendrés et rejetés.
 - ▷ Facteurs gauches : tirer n pas uniformément dans $\{\nearrow, \searrow, \rightarrow\}$.
 $O(n\sqrt{n})$: on rejette \sqrt{n} chemins en moyenne, car $f_n \sim 3^n \sqrt{\frac{3}{\pi(n+1)}}$.

Génération par **rejet anticipé** :

- essayer de rejeter **avant** d'avoir engendré le chemin complet.
 - ▷ Facteurs gauches : rejet dès que la hauteur devient négative (rejet florentin). Il faut tirer en moyenne $2n$ lettres pour engendrer un facteur de taille n , donc la complexité moyenne est en $O(n)$.

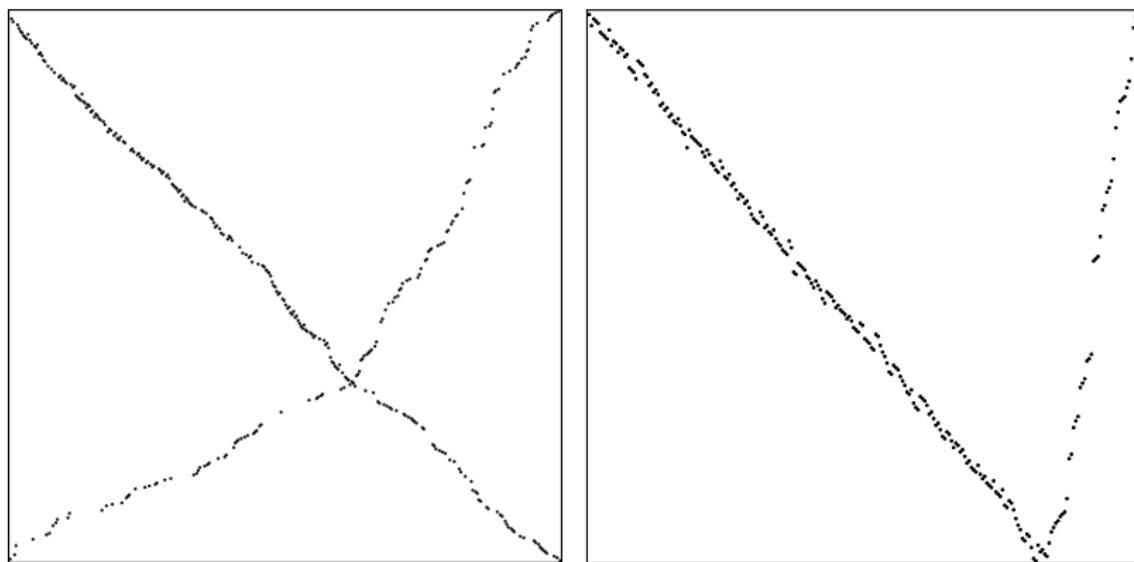
Amélioration : rejet avec “récupération”, rejet probabiliste (cf. travaux d'A. Bacher, O. Bodini, A. Jacquot, A. Sportiello, ...)



- 1 Contexte
- 2 Méthode Récursive
- 3 Interlude : créer automatiquement un générateur à partir d'une spécification
- 4 Méthode de Boltzmann
- 5 Questions d'implémentation

Classes de permutation à motifs exclus

Permutations aléatoires dans $\mathcal{C} = \text{Av}(2413, 3142, 2143, 34512)$ et $\mathcal{C} = \text{Av}(2413, 1243, 2341, 41352, 531642)$

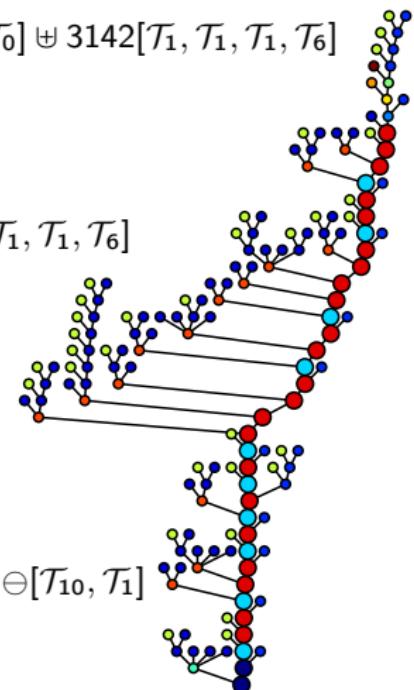


Scaling limits of permutation classes with a finite specification : a dichotomy,
F. Bassino, M. Bouvel, V. Féray, L. Gerin, M. Maazoun, A. Pierrot, 2022.

Classes de permutation à motifs exclus

Spécification pour : $Av(2413, 1243, 2341, 531642, 41352)$

$$\left\{ \begin{array}{l} \mathcal{T}_0 = \{\bullet\} \uplus \oplus[\mathcal{T}_1, \mathcal{T}_2] \uplus \oplus[\mathcal{T}_1, \mathcal{T}_3] \uplus \oplus[\mathcal{T}_4, \mathcal{T}_2] \uplus \ominus[\mathcal{T}_5, \mathcal{T}_0] \uplus 3142[\mathcal{T}_1, \mathcal{T}_1, \mathcal{T}_1, \mathcal{T}_6] \\ \mathcal{T}_1 = \{\bullet\} \uplus \ominus[\mathcal{T}_7, \mathcal{T}_1] \\ \mathcal{T}_2 = \{\bullet\} \uplus \oplus[\mathcal{T}_7, \mathcal{T}_2] \\ \mathcal{T}_3 = \oplus[\mathcal{T}_8, \mathcal{T}_2] \uplus \ominus[\mathcal{T}_9, \mathcal{T}_6] \\ \mathcal{T}_4 = \ominus[\mathcal{T}_{10}, \mathcal{T}_{11}] \uplus \ominus[\mathcal{T}_{10}, \mathcal{T}_1] \uplus \ominus[\mathcal{T}_7, \mathcal{T}_{11}] \uplus 3142[\mathcal{T}_1, \mathcal{T}_1, \mathcal{T}_1, \mathcal{T}_6] \\ \mathcal{T}_5 = \{\bullet\} \uplus \oplus[\mathcal{T}_1, \mathcal{T}_1] \uplus 3142[\mathcal{T}_1, \mathcal{T}_1, \mathcal{T}_1, \mathcal{T}_1] \\ \mathcal{T}_6 = \{\bullet\} \uplus \oplus[\mathcal{T}_{12}, \mathcal{T}_2] \uplus \ominus[\mathcal{T}_9, \mathcal{T}_6] \\ \mathcal{T}_7 = \{\bullet\} \\ \mathcal{T}_8 = \ominus[\mathcal{T}_9, \mathcal{T}_6] \\ \mathcal{T}_9 = \{\bullet\} \uplus \oplus[\mathcal{T}_1, \mathcal{T}_7] \\ \mathcal{T}_{10} = \oplus[\mathcal{T}_1, \mathcal{T}_1] \uplus 3142[\mathcal{T}_1, \mathcal{T}_1, \mathcal{T}_1, \mathcal{T}_1] \\ \mathcal{T}_{11} = \oplus[\mathcal{T}_1, \mathcal{T}_2] \uplus \oplus[\mathcal{T}_1, \mathcal{T}_3] \uplus \oplus[\mathcal{T}_4, \mathcal{T}_2] \uplus \ominus[\mathcal{T}_{10}, \mathcal{T}_{11}] \uplus \ominus[\mathcal{T}_{10}, \mathcal{T}_1] \\ \quad \uplus \ominus[\mathcal{T}_7, \mathcal{T}_{11}] \uplus 3142[\mathcal{T}_1, \mathcal{T}_1, \mathcal{T}_1, \mathcal{T}_6] \\ \mathcal{T}_{12} = \{\bullet\} \uplus \ominus[\mathcal{T}_9, \mathcal{T}_6] \end{array} \right.$$



Comment savoir si la spécification est "valide" ?

Spécification bien définie ?

$$\mathcal{Y}_1 = \text{Seq}(\mathcal{E}) \text{ X} \quad \mathcal{Y}_1 = \text{Seq}(\text{Cyc}(\mathcal{Z})) \checkmark \quad \mathcal{Y}_1 = \text{Seq}(\text{Seq}(\mathcal{Z})) \text{ X}$$

"Itérativement" bien définie ?

$$\begin{array}{lll} \mathcal{Y}_1 = \text{Seq}(\mathcal{Y}_2) & \mathcal{Y}_1 = \text{Seq}(\mathcal{Y}_2) & \mathcal{Y}_1 = \text{Seq}(\mathcal{Y}_2) \\ \mathcal{Y}_2 = \mathcal{Z} + \mathcal{Z}\mathcal{Y}_2 \checkmark & \mathcal{Y}_2 = \mathcal{E} + \mathcal{Z}\mathcal{Y}_2 \text{ X} & \mathcal{Y}_2 = \mathcal{Z} + \mathcal{Y}_3 + \mathcal{Y}_4 \\ & & \mathcal{Y}_3 = \mathcal{Z} \text{ Seq}(\mathcal{Z}) \\ & & \mathcal{Y}_4 = \mathcal{Y}_1 + \mathcal{Y}_1\mathcal{Y}_4 \text{ X} \end{array}$$

Le cas des 0 ?

$$\mathcal{Y} = \mathcal{Z} \mathcal{Y} \rightarrow \mathcal{Y} = 0 \checkmark \quad \text{Les 0 sont visibles dans } \mathcal{H}^m(\mathcal{Z}, 0)$$

Un nombre fini de structures de chaque taille ?

$$\begin{array}{lll} \mathcal{Y}_1 = \mathcal{Z} + \mathcal{Y}_1 \text{ X} & \mathcal{Y}_1 = \mathcal{Z} + \mathcal{Y}_1^2 \checkmark & \mathcal{Y}_1 = \mathcal{Z} + \mathcal{Y}_2 \\ & & \mathcal{Y}_2 = \mathcal{Z} + \mathcal{Y}_1 \mathcal{Y}_2 \checkmark \end{array}$$

Algorithme : spécification est bien fondée ?

Entrée : $\mathcal{Y} = \mathcal{H}(\mathcal{Z}, \mathcal{Y})$ une spécification combinatoire à m équations,
en forme normale.

$\mathcal{U}_{1:m} \leftarrow (0, \dots, 0)$

Faire m fois

pour chaque $\mathcal{H}_i = \mathcal{F}(\mathcal{Y}_j)$ # $i = 1..m$

renvoyer FAUX si $\mathcal{F} \in \{\text{Set}, \text{Seq}, \text{Cyc}\}$ et $\mathcal{E} \in \mathcal{U}_j$

$\mathcal{V}_i \leftarrow \mathcal{H}_i(0, \mathcal{U})$

$\mathcal{U} \leftarrow \mathcal{V}$ # structures de taille 0

calculer $\mathcal{W} := \mathcal{H}^m(\mathcal{Z}, \mathbf{0})$ # détection des 0

retirer de \mathcal{H} les coordonées corresp. à des 0 dans \mathcal{W}

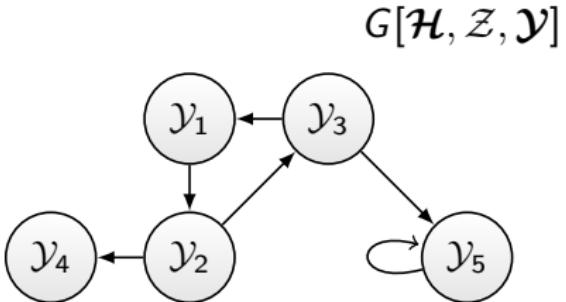
renvoyer VRAI si $G[\mathcal{H}, \mathbf{0}, \mathcal{U}]$ n'a pas de cycle, FAUX sinon



Graphe de dépendance de $\mathcal{Y} = \mathcal{H}(\mathcal{Z}, \mathcal{Y})$ pour $\mathcal{Z} = \mathbf{0}$ et $\mathcal{Y} = \mathcal{U}$
= matrice Jacobienne du système évaluée en $\mathbf{0}, \mathcal{U}$.

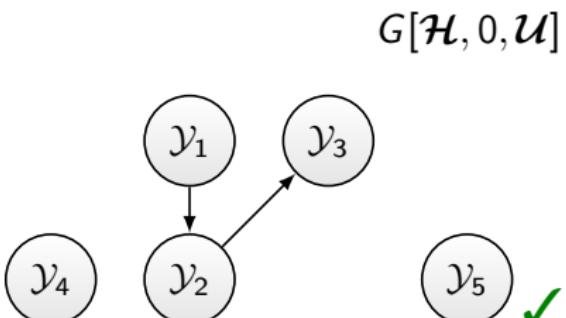
Graphe de dépendance et matrice Jacobienne

$$\begin{cases} \mathcal{Y}_1 = \mathcal{Z} + \mathcal{Y}_2 \\ \mathcal{Y}_2 = \mathcal{Y}_4 \mathcal{Y}_3 \\ \mathcal{Y}_3 = \mathcal{Y}_5 \mathcal{Y}_1 \\ \mathcal{Y}_4 = \text{Seq}(\mathcal{Z}) \\ \mathcal{Y}_5 = \mathcal{Z} + \mathcal{Y}_5^2 \end{cases}$$



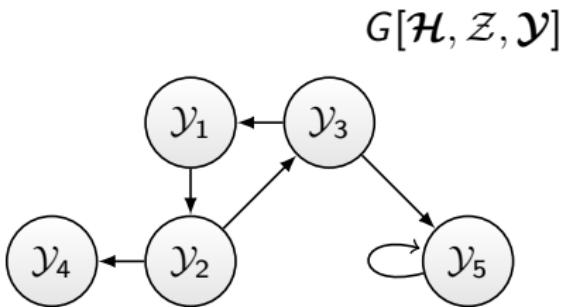
$$\mathcal{U} = (0, 0, 0, 1, 0)$$

$$\begin{pmatrix} 0 & \mathcal{E} & 0 & 0 & 0 \\ 0 & 0 & \mathcal{Y}_4 & \mathcal{Y}_3 & 0 \\ \mathcal{Y}_5 & 0 & 0 & 0 & \mathcal{Y}_1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2\mathcal{Y}_5 \end{pmatrix}$$



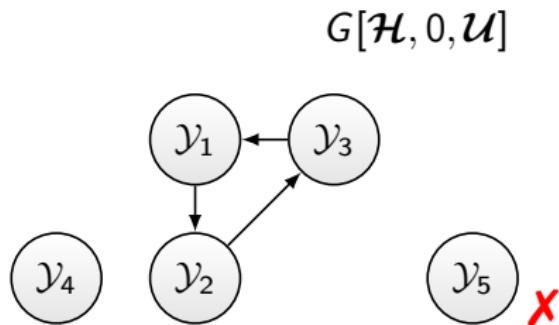
Graphe de dépendance et matrice Jacobienne

$$\begin{cases} \mathcal{Y}_1 = \mathcal{Z} + \mathcal{Y}_2 \\ \mathcal{Y}_2 = \mathcal{Y}_4 \mathcal{Y}_3 \\ \mathcal{Y}_3 = \mathcal{Y}_5 \mathcal{Y}_1 \\ \mathcal{Y}_4 = \text{Seq}(\mathcal{Z}) \\ \mathcal{Y}_5 = \mathcal{E} + \mathcal{Z} \mathcal{Y}_5^2 \end{cases}$$



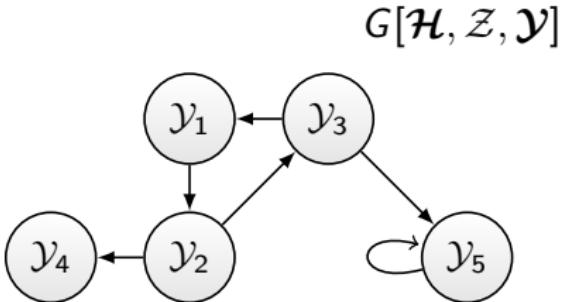
$$\mathcal{U} = (0, 0, 0, 1, 1)$$

$$\begin{pmatrix} 0 & \mathcal{E} & 0 & 0 & 0 \\ 0 & 0 & \mathcal{Y}_4 & \mathcal{Y}_3 & 0 \\ \mathcal{Y}_5 & 0 & 0 & 0 & \mathcal{Y}_1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2\mathcal{Z}\mathcal{Y}_5 \end{pmatrix}$$



Graphe de dépendance et matrice Jacobienne

$$\begin{cases} \mathcal{Y}_1 = \mathcal{Z} + \mathcal{Y}_2 \\ \mathcal{Y}_2 = \mathcal{Y}_4 \mathcal{Y}_3^2 \\ \mathcal{Y}_3 = \mathcal{Y}_5 \mathcal{Y}_1 \\ \mathcal{Y}_4 = \text{Seq}(\mathcal{Z}) \\ \mathcal{Y}_5 = \mathcal{E} + \mathcal{Z} \mathcal{Y}_5^2 \end{cases}$$



$$\mathcal{U} = (0, 0, 0, 1, 1)$$

$$\left(\begin{array}{ccccc} 0 & \mathcal{E} & 0 & 0 & 0 \\ 0 & 0 & 2\mathcal{Y}_4\mathcal{Y}_3 & \mathcal{Y}_3^2 & 0 \\ \mathcal{Y}_5 & 0 & 0 & 0 & \mathcal{Y}_1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2\mathcal{Z}\mathcal{Y}_5 \end{array} \right) \quad G[\mathcal{H}, 0, \mathcal{U}]$$

```

graph TD
    Y1((Y1)) --> Y2((Y2))
    Y3((Y3)) --> Y1
    Y3 --> Y2
    Y3 --> Y5((Y5))
    Y5 --> Y2
  
```

Librairie combstruct en Maple (démo, s'il reste du temps...)

Gaïa : a package for the random generation of combinatorial structures
P. Zimmermann, 1994

MuPad-Combinat (Sage) : decomposableObjects

<https://mupad-combinat.sourceforge.net/doc/en/combinat/decomposableObjects.html>

CS : a MuPAD package for counting and randomly generating combinatorial structures, A. Denise, I. Dutour, P. Zimmermann, 1998

MuPAD-Combinat, an open-source package for research in algebraic combinatorics, F. Hivert, N. Thiéry, 2003

Fin de la première partie

Les *notebooks* en Python des exercices (et du cours) sont ici :

<https://github.com/CarinePivoteau/Alea2023Notebooks>

Pour installer sur votre machine :

```
pip3 install --upgrade pip  
pip3 install jupyter
```

Pour utiliser les *notebooks*, cloner le dépôt ou télécharger son archive. Ensuite, en se plaçant dans le dossier qui les contient, taper la commande :

```
jupyter-notebook
```

- 1 Contexte
- 2 Méthode Récursive
- 3 Interlude : créer automatiquement un générateur à partir d'une spécification
- 4 Méthode de Boltzmann
- 5 Questions d'implémentation

1 Contexte

2 Méthode Récursive

3 Interlude : créer automatiquement un générateur à partir d'une spécification

4 Méthode de Boltzmann

- Un mot sur les arbres de Bienaymé-Galton-Watson
- Le principe
- Algorithmes récursifs de génération (sans symétrie)
- Exemples de générateur "libres"
- Générateurs pour les constructions avec symétries
- Exemples et applications

5 Questions d'implémentation

Arbre de Bienaym  -Galton-Watson

Arbre planaire enracin   al  atoire d  fini par une loi de probabilit  s sur \mathbb{N} : $(\pi_k)_{k=0}^{\infty}$, telle que π_k est la probabilit   pour un n  ud d'avoir k fils.

Remarque : le nombre de fils ne d  pend ni de la position du n  ud dans l'arbre ni du nombre de fils des autres n  uds.

L'esp  rance du nombre de fils d'un n  ud est $m = \sum_{n=0}^{\infty} k\pi_k$.

Si $m = 1$, alors :

- on est dans le cas **critique**,
- l'esp  rance de la taille de l'arbre est **infinie**,
- la probabilit   que la taille de l'arbre engendr   soit **finie** est 1.

(Arbres et processus de Galton-Watson, J. Neveu, 1986)

(Simply generated trees, conditioned Galton-Watson trees, random allocations and condensation, S. Janson, 2012)

Exemple des arbres binaires

Un arbre binaire aléatoire est composé :

- d'un nœud racine,
- d'un fils gauche : avec proba. $1/2$ une feuille ou un arbre aléatoire,
- d'un fils droit : avec proba. $1/2$ une feuille ou un arbre aléatoire.

Pour un arbre T quelconque de taille n (n nœuds internes et $n+1$ feuilles), chaque nœud ou feuille est choisi avec probabilité $\frac{1}{2}$, sauf la racine. La probabilité d'engendrer T est $\mathbb{P}(T) = \left(\frac{1}{2}\right)^{2n} = \left(\frac{1}{4}\right)^n$.

Cette probabilité ne dépend que de la taille n de l'arbre, on a donc des **arbres uniformes pour chaque taille**.

On peut obtenir un générateur aléatoire uniforme d'arbres binaires en conditionnant par la taille et en utilisant le **rejet anticipé**.

Code en python

```
def gen_free(max):
    size = 0
    def random_GW():
        nonlocal size # compteur global
        size += 1
        if size > max: # rejet anticipé
            raise Exception("Too large")

        left = random_GW() if flip_a_coin() else []
        right = random_GW() if flip_a_coin() else []
        return ['z', left, right]

    try:
        return random_GW(), size
    except:
        return None, 0

def gen_approx(mini, maxi):
    while True:
        tree, size = gen_free(n)
        if mini <= size <= maxi:
            return tree
```

Remarque : pour le rejet anticipé, on peut aussi générer en largeur, pour éviter d'avoir à dépiler le calcul.

1 Contexte

2 Méthode Récursive

3 Interlude : créer automatiquement un générateur à partir d'une spécification

4 Méthode de Boltzmann

- Un mot sur les arbres de Bienaymé-Galton-Watson
- Le principe
- Algorithmes récursifs de génération (sans symétrie)
- Exemples de générateur "libres"
- Générateurs pour les constructions avec symétries
- Exemples et applications

5 Questions d'implémentation

Modèle de Boltzmann

La distribution des tailles des structures s'étale sur l'ensemble d'une classe combinatoire mais reste **uniforme** pour chaque sous-classe de structures de **même taille**.

Definition (Duchon, Flajolet, Louchard, Schaeffer – 2004)

Pour $x < \rho_{\mathcal{C}}$, le **modèle de Boltzmann** assigne à tout objet c de la classe \mathcal{C} la probabilité :

$$\mathbb{P}_x(c) = \frac{x^{|c|}}{C(x)} \text{ (non étiqueté)} \quad \text{ou} \quad \mathbb{P}_x(c) = \frac{x^{|c|}}{|c|! \hat{C}(x)} \text{ (étiqueté)}$$

Un générateur de Boltzmann **GenB**(\mathcal{C}, x) pour la classe \mathcal{C} est un algorithme qui produit des objets de \mathcal{C} suivant ce modèle.

- **structures de même taille \leftrightarrow même probabilité**
- génération en **taille approchée** (paramètre de contrôle).
- de très grandes structures peuvent être engendrées.

Théorème (Générateurs étiquetés – DuFILoSc04)

Pour toute classe \mathcal{C} étiquetée définie par une spécification combinatoire constructible et étant donné un oracle pour cette spécification, le générateur de Boltzmann libre $\text{GenB}(\mathcal{C}, x)$ opère en temps linéaire par rapport à la taille de la structure produite.

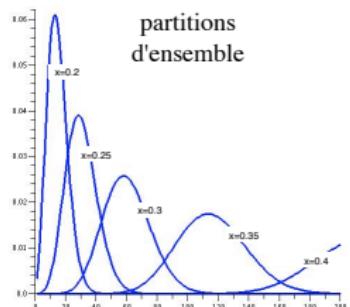
- **Constructible** : bien fondée et qui utilise les constructions “usuelles” (sauf PSet)
- **Oracle** : évaluation numérique des séries génératrices en x .
- **Libre** : qui engendre des structures de n’importe quelle taille...
 - ▷ viser la taille n : choisir x tel que l’espérance de la taille est n .

$$\mathbb{P}_x(N = n) = \frac{c_n x^n}{C(x)} \quad \text{donc} \quad \mathbb{E}_x(N) = x \frac{C'(x)}{C(x)}$$

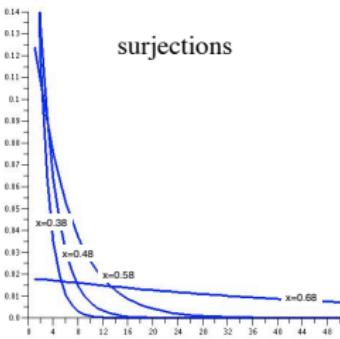
- Générateur en taille approchée ou exacte : utiliser du **rejet**.

Génération en taille approchée/exacte

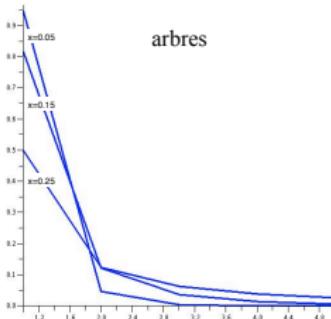
C'est la distribution des tailles qui détermine le coût du rejet.



$$C(x) = e^{(e^x - 1)}$$



$$C(x) = \frac{1}{2 - e^x}$$



$$C(x) = \frac{1 - \sqrt{1 - 4x}}{2}$$

Théorème (Complexités – DuFILoSc04)

type de distribution	concentrée	plate	piquée
taille approchée	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$o(n^2)$
taille exacte	$o(n^2)$	$o(n^2)$	–

▷ pointage
gen. singulier

1 Contexte

2 Méthode Récursive

3 Interlude : créer automatiquement un générateur à partir d'une spécification

4 Méthode de Boltzmann

- Un mot sur les arbres de Bienaymé-Galton-Watson
- Le principe
- Algorithmes récursifs de génération (sans symétrie)
- Exemples de générateur “libres”
- Générateurs pour les constructions avec symétries
- Exemples et applications

5 Questions d'implémentation

Union disjointe $\mathcal{C} = \mathcal{A} + \mathcal{B}$

Choisir d'engendrer une structure de \mathcal{A} avec probabilité $A(x)/C(x)$ ou une structure dans \mathcal{B} sinon.

GenB($\mathcal{A} + \mathcal{B}, x$)

$U \leftarrow \text{random}()$

si $U < A(x)/C(x)$ **alors**

renvoyer **GenB**(\mathcal{A}, x)

sinon renvoyer **GenB**(\mathcal{B}, x)

Uniformité à taille fixée : une structure aléatoire γ de \mathcal{C} , de taille n , est engendrée avec probabilité

$$\mathbb{P}_x(\gamma \in \mathcal{A}) = \frac{A(x)}{C(x)} \cdot \frac{x^{|\gamma|}}{A(x)} \quad \text{ou} \quad \mathbb{P}_x(\gamma \in \mathcal{B}) = \frac{B(x)}{C(x)} \cdot \frac{x^{|\gamma|}}{B(x)} \quad \text{donc} \quad \frac{x^n}{C(x)}$$

ou

$$\mathbb{P}_x(\gamma \in \mathcal{A}) = \frac{\hat{A}(x)}{C(x)} \cdot \frac{x^{|\gamma|}}{|\gamma|! \hat{A}(x)}, \quad \mathbb{P}_x(\gamma \in \mathcal{B}) = \frac{\hat{B}(x)}{C(x)} \cdot \frac{x^{|\gamma|}}{|\gamma|! \hat{B}(x)} \quad \text{donc} \quad \frac{x^n}{n! C(x)}$$

Produit Cartésien $\mathcal{C} = \mathcal{A} \times \mathcal{B}$

Former un couple (α, β) , avec α et β engendrés par des appels **indépendants** aux générateurs $\text{GenB}(\mathcal{A}, x)$ et $\text{GenB}(\mathcal{B}, x)$.

$\text{GenB}(\mathcal{A} \times \mathcal{B}, x)$

| **renvoyer** $(\text{GenB}(\mathcal{A}, x), \text{GenB}(\mathcal{B}, x))$

Étiquetage : après coup, pour éviter de redistribuer les étiquettes.

Uniformité : la taille n d'une structure γ de \mathcal{C} est la somme des tailles a de α et b de β . La probabilité d'engendrer γ est

$$\mathbb{P}_x(\gamma) = \frac{x^a}{A(x)} \cdot \frac{x^b}{B(x)} = \frac{x^n}{C(x)}$$

ou

$$\mathbb{P}_x(\gamma) = \frac{x^a}{a! \hat{A}(x)} \cdot \frac{x^b}{b! \hat{B}(x)} \cdot \binom{a! b!}{n!} = \frac{x^n}{n! \hat{C}(x)}$$

Complexité : pas de coût additionnel pour choisir les tailles a et b .

Séquence $\mathcal{C} = \text{Seq}(\mathcal{A}) \Rightarrow \mathcal{C} = \mathcal{E} + \mathcal{A} \times \mathcal{C}$

En utilisant les algorithmes précédents, on obtient le générateur :

```
GenB(Seq( $\mathcal{A}$ ),  $x$ )
   $U \leftarrow \text{random}()$ 
  si  $U < A(x)$  alors
    renvoyer (GenB( $\mathcal{A}$ ,  $x$ ) , GenB(Seq( $\mathcal{A}$ ),  $x$ ) )
  sinon renvoyer  $\mathcal{E}$ 
```

▷ tirages de Bernoulli de paramètre $A(x)$ jusqu'à obtenir un échec. Le nombre de \mathcal{A} -structures engendrées suit une loi géométrique de paramètre $\lambda = A(x)$:

$$\mathbb{P}(X = k) = (1 - \lambda)\lambda^k.$$

On peut donc réécrire le générateur, en tirant le nombre d'éléments de la séquence suivant la loi géométrique $\text{Geom}(A(x))$:

```
GenB(Seq( $\mathcal{A}$ ),  $x$ )
   $k \leftarrow \text{Geom}(A(x))$ 
  renvoyer le  $k$ -uplet (GenB( $\mathcal{A}$ ,  $x$ ) , . . . , GenB( $\mathcal{A}$ ,  $x$ ))
```

Ensembles et cycles (étiquetés uniquement)

On rappelle que la loi de Poisson de paramètre λ est définie par :

$$\mathbb{P}(X = k) = e^{-\lambda} \frac{\lambda^k}{k!}$$

La loi logarithmique de paramètre λ est définie par :

$$\mathbb{P}(X = k) = \frac{1}{\log(1 - \lambda)^{-1}} \frac{\lambda^k}{k}$$

Les générateurs pour les ensembles et les cycles de \mathcal{A} -structures :

GenB(Set(\mathcal{A}), x)

$k \leftarrow \text{Poiss}(\mathcal{A}(x))$

renvoyer le k -uplet $\{\mathbf{GenB}(\mathcal{A}, x), \dots, \mathbf{GenB}(\mathcal{A}, x)\}$

GenB(Cyc(\mathcal{A}), x)

$k \leftarrow \text{Loga}(\mathcal{A}(x))$

renvoyer le k -uplet $(\mathbf{GenB}(\mathcal{A}, x), \dots, \mathbf{GenB}(\mathcal{A}, x))$

	s.g. ordinaires (non étiqueté)	s.g. exponentielles (étiqueté)	
1 ou \mathcal{Z}	1 ou z	1 ou z	
$\mathcal{A} + \mathcal{B}$	$A(z) + B(z)$	$A(z) + B(z)$	→ loi de Bernoulli
$\mathcal{A} \times \mathcal{B}$	$A(z) \times B(z)$	$A(z) \times B(z)$	→ pas de choix
$\text{Seq}(\mathcal{A})$	$\frac{1}{1 - A(z)}$	$\frac{1}{1 - A(z)}$	→ loi géométrique ▷ $\mathbb{P}(X = k) = (1 - \lambda)\lambda^k$
$\text{MSet}(\mathcal{A})$...	$\exp(A(z))$	→ loi de Poisson ▷ $\mathbb{P}(X = k) = e^{-\lambda} \frac{\lambda^k}{k!}$
$\text{PSet}(\mathcal{A})$...	—	
$\text{Cyc}(\mathcal{A})$...	$\log \frac{1}{1 - A(z)}$	→ loi logarithmique ▷ $\mathbb{P}(X = k) = \frac{1}{\log(1-\lambda)^{-1}} \frac{\lambda^k}{k}$

Lois usuelles : par découpage d'intervalle

Schéma itératif générique :

- découper l'intervalle $[0, 1]$ selon les probabilités $p_k = \mathbb{P}(K = k)$,
- puis tirer un réel aléatoire U dans $[0, 1]$,
- et chercher séquentiellement dans quel sous-intervalle se trouve U .

GenLoi(x, k, p_k) # $k = 0$ sauf pour Poiss $_{\geq 1}$

$U \leftarrow \text{random}()$

$S \leftarrow p_k$

tant que $S < U$ **faire**

$S \leftarrow S + p_k$

$k \leftarrow k + 1$

renvoyer k

Geom(λ)	Poiss(λ)	Poiss $_{\geq 1}(\lambda)$	Loga(λ)
$p_k = (1 - \lambda)\lambda^k$	$p_k = e^{-\lambda} \frac{\lambda^k}{k!}$	$p_k = e^{-\lambda} \frac{\lambda^k}{k!}$	$p_k = \frac{1}{\log(1-\lambda)^{-1}} \frac{\lambda^k}{k}$
$p_0 = (1 - \lambda)$	$p_0 = e^{-\lambda}$	$p_1 = \frac{\lambda}{e^\lambda - 1}$	$p_0 = 1 / (\log(1-\lambda)^{-1})$
$p_{k+1} = \lambda p_k$	$p_{k+1} = \lambda p_k \frac{1}{k+1}$	$p_{k+1} = \lambda p_k \frac{1}{k+1}$	$p_{k+1} = \lambda p_k \frac{k}{k+1}$

1 Contexte

2 Méthode Récursive

3 Interlude : créer automatiquement un générateur à partir d'une spécification

4 Méthode de Boltzmann

- Un mot sur les arbres de Bienaymé-Galton-Watson
- Le principe
- Algorithmes récursifs de génération (sans symétrie)
- Exemples de générateur "libres"
- Générateurs pour les constructions avec symétries
- Exemples et applications

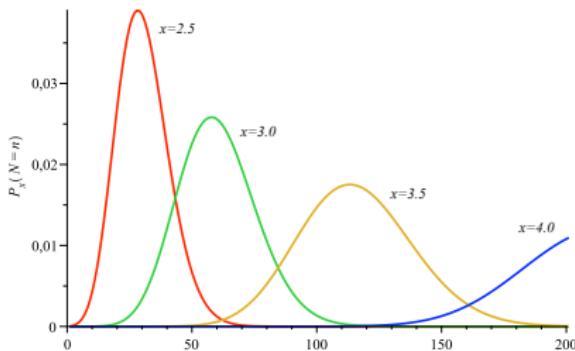
5 Questions d'implémentation

Partitions d'ensemble (spec. étiquetée)

Les partitions d'ensemble sont définies par la spécification :

$$\mathcal{P} = \text{Set}(\text{Set}_{\geq 1}(\mathcal{Z}))$$

- La série génératrice exponentielle associée est $L(x) = e^{e^x - 1}$.
- La probabilité pour une partition p d'être tirée sous modèle de Boltzmann est $\mathbb{P}_x(p) = x^{|p|} e^{1-e^x}$.
- L'espérance de la taille des partitions engendrées est $\mathbb{E}_x(N) = xe^x$.



Distribution des tailles des partitions pour différentes valeurs de x . Pour la courbe jaune ($x = 3.5$), l'espérance de la taille est $\mathbb{E}_{3.5}(N) = 115.9$.

Partitions d'ensemble - suite

Le générateur **GenB**(\mathcal{P}, x) pour les partitions d'ensemble se déduit de la spécification et des règles précédentes :

GenB(\mathcal{P}, x)

$k \leftarrow \text{Poiss}(e^x - 1)$

pour i **de** 1 **à** k **faire**

$\ell_i \leftarrow \text{Poiss}_{\geq 1}(x)$

renvoyer le k -uplet $\{\ell_1, \ell_2, \dots, \ell_k\}$

Résultat : une liste des tailles des ensembles qui forment la partition p .

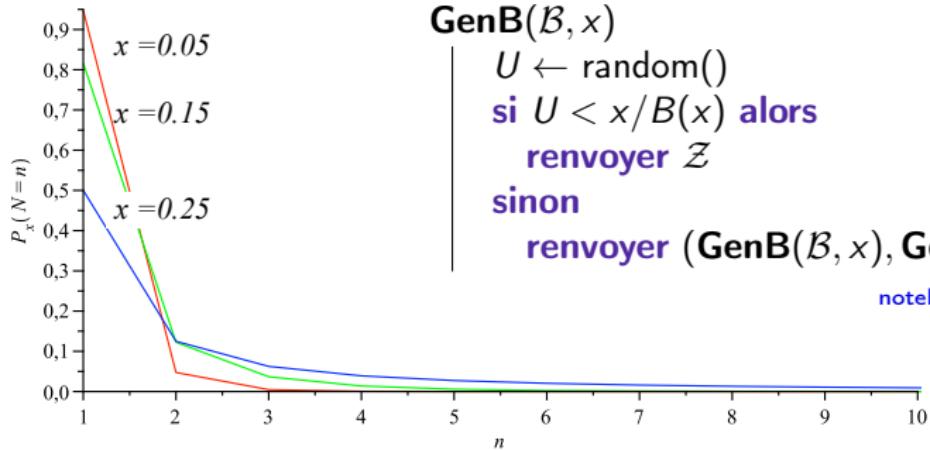
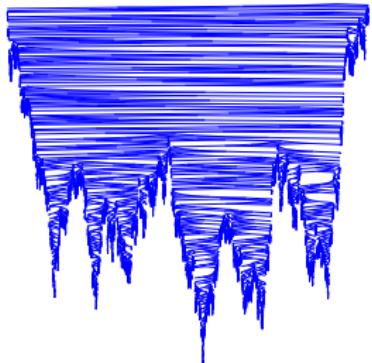
Pour obtenir p , il suffit de tirer une permutation aléatoire dont la longueur est le nombre d'éléments de p (la somme des ℓ_i) et d'étiqueter les atomes de p dans l'ordre de cette permutation.

$$\left\{ \underbrace{\{\bullet\bullet\bullet\bullet\}}_{\ell_1} \underbrace{\{\bullet\bullet\}}_{\ell_2} \cdots \underbrace{\{\bullet\bullet\bullet\bullet\bullet\}}_{\ell_k} \right\} \rightarrow \left\{ \{2938\} \{114\} \cdots \{12101675\} \right\}$$

Programme : en TP

Générateur d'arbres binaires planaires

$$\mathcal{B} = \mathcal{Z} + \mathcal{B} \times \mathcal{B}$$
$$B(z) = z + B(z)^2 = \frac{1 - \sqrt{1 - 4z}}{2}$$



GenB(\mathcal{B}, x)

$U \leftarrow \text{random}()$

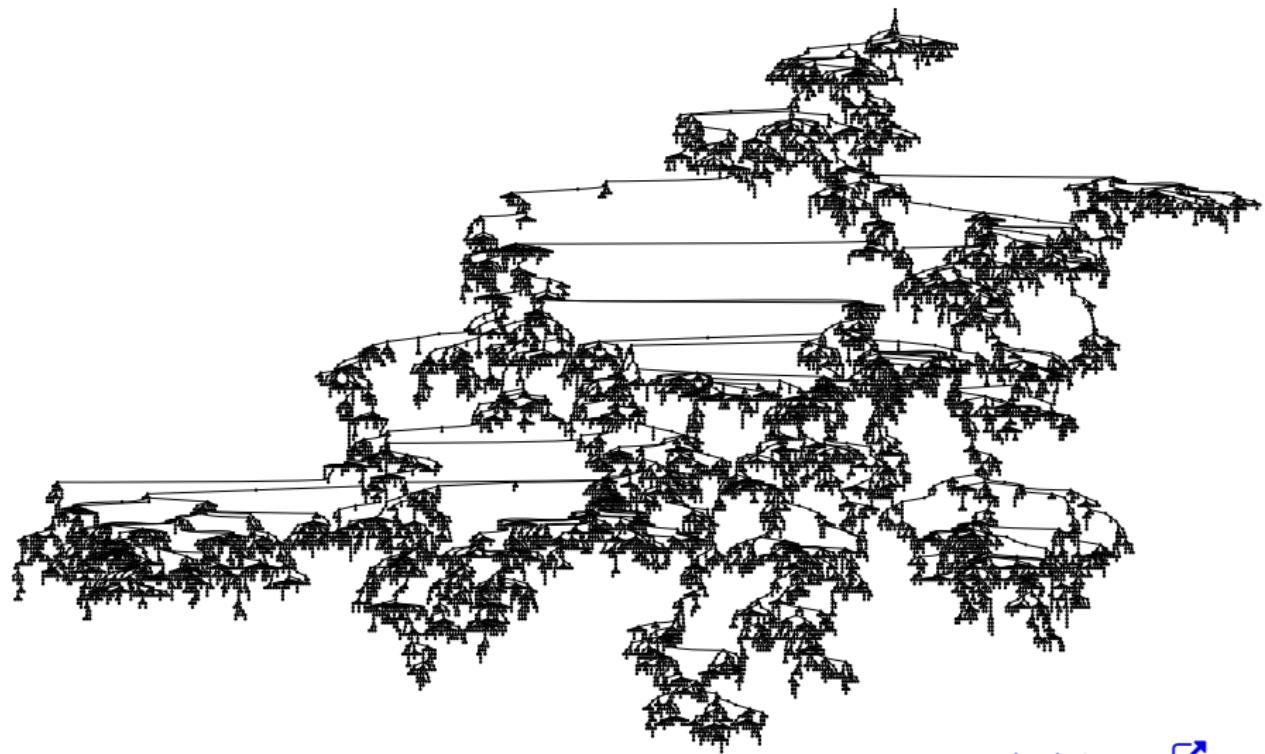
si $U < x/B(x)$ **alors**

renvoyer \mathcal{Z}

sinon

renvoyer (**GenB(\mathcal{B}, x)**, **GenB(\mathcal{B}, x)**)

notebook Jupyter ↗

[notebook Jupyter](#) ↗

1 Contexte

2 Méthode Récursive

3 Interlude : créer automatiquement un générateur à partir d'une spécification

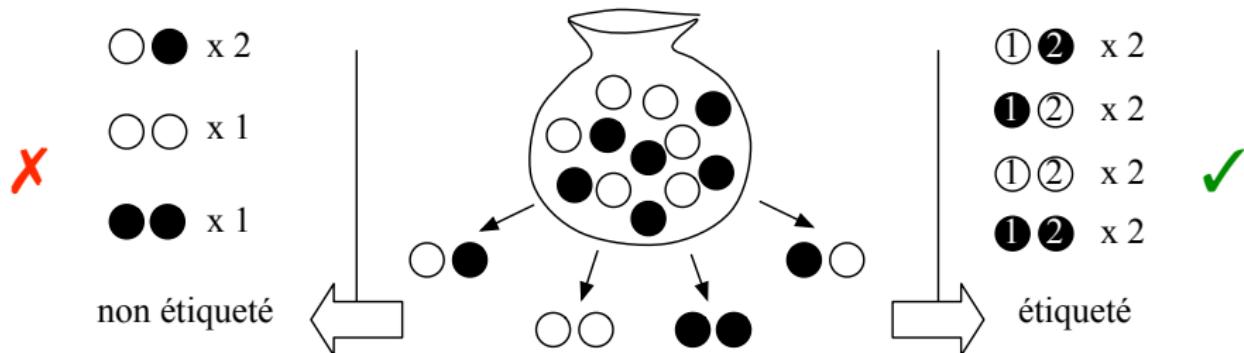
4 Méthode de Boltzmann

- Un mot sur les arbres de Bienaymé-Galton-Watson
- Le principe
- Algorithmes récursifs de génération (sans symétrie)
- Exemples de générateur "libres"
- **Générateurs pour les constructions avec symétries**
- Exemples et applications

5 Questions d'implémentation

Le problème des symétries

Comment engendrer une paire non ordonnée d'atomes colorés ?
Autrement dit un élément de $MSet_2(\mathcal{Z} + \mathcal{Z})\dots$



$$MSet_2(\mathcal{A}) = \frac{1}{2} A(z)^2 + \frac{1}{2} A(z^2)$$

\uparrow
série génératrice de $\Delta^{(2)} \mathcal{A}$.

Opérateurs de Pólya

	construction	s.g. ordinaire (non étiquetée)	s.g. exponentielle (étiquetée)
\mathcal{E} ou atome	1 ou \mathcal{Z}	1 ou z	1 ou z
Union	$\mathcal{A} + \mathcal{B}$	$A(z) + B(z)$	$A(z) + B(z)$
Produit	$\mathcal{A} \times \mathcal{B}$	$A(z) \times B(z)$	$A(z) \times B(z)$
Séquence	$\text{Seq}(\mathcal{A})$	$\frac{1}{1 - A(z)}$	$\frac{1}{1 - A(z)}$
Ensemble	$\text{Set/PSet}(\mathcal{A})$	$\exp\left(\sum_{k=1}^{\infty} \frac{(-1)^{k-1}}{k} A(z^k)\right)$	$\exp(A(z))$
Multi-ensemble	$\text{MSet}(\mathcal{A})$	$\exp\left(\sum_{k=1}^{\infty} \frac{1}{k} A(z^k)\right)$	-
Cycle	$\text{Cyc}(\mathcal{A})$	$\sum_{k=1}^{\infty} \frac{\varphi(k)}{k} \log \frac{1}{1 - A(z^k)}$	$\log \frac{1}{1 - A(z)}$

Théorème (Générateurs non étiquetés)

Pour toute classe \mathcal{C} non étiquetée spécifiée (éventuellement récursivement) à partir des constructions suivantes :

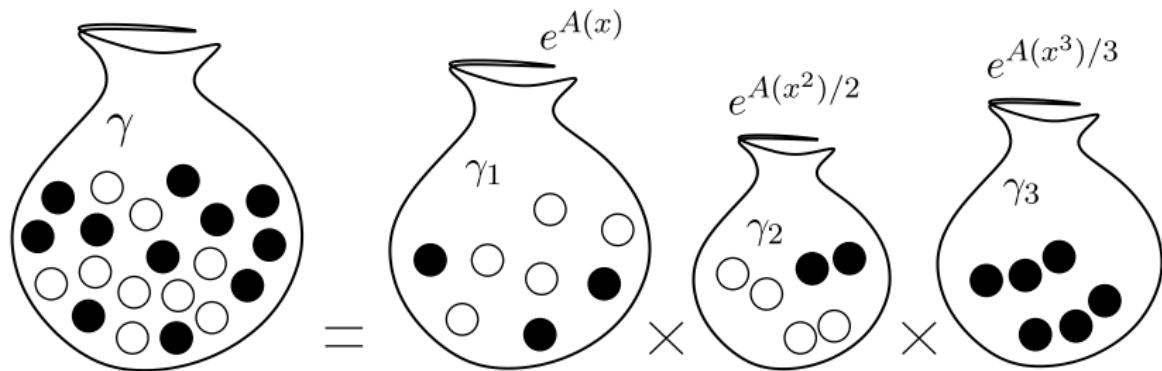
$\varepsilon, \mathcal{Z}, +, \times, \text{Seq}, \text{Seq}_k, \text{MSet}, \text{MSet}_k, \text{Cyc}, \text{Cyc}_k,$

le générateur de Boltzmann libre $\text{GenB}(C, x)$ opère en temps linéaire par rapport à la taille de la structure produite.

- Les variantes des algorithmes pour les contraintes de cardinalité de type $= k$ sont adaptées des algorithmes généraux.
- Pour les contraintes de cardinalité de type $\leq k$ ont fait des combinaisons des précédentes.
- Pour les contraintes de cardinalité de type $\geq k$, on utilise du rejet.

Boltzmann sampling of unlabelled structures, P. Flajolet, É. Fusy et C. Pivoteau, 2007

$$C(x) = \exp\left(\sum_{k=1}^{\infty} \frac{1}{k} A(z^k)\right) = \prod_{k=1}^{\infty} \exp\left(\frac{1}{k} A(x^k)\right)$$



- Choisir k tel que : $\mathbb{P}(K = k) = \left(\prod_{j \leq k} \exp(A(x^j)/j) \right) / C(x)$
- Pour chaque $\exp(\frac{1}{i} A(x^i))$
 - Choisir p suivant une loi de Poisson de paramètre $A(x^i)/i$.
 - Engendrer indépendamment p éléments de $\Delta^{(i)}\mathcal{A}$.

Engendrer des multi-ensembles - suite

GenB(MSet(\mathcal{A}), x)

$M \leftarrow \emptyset$

$k \leftarrow \text{IndiceMaxMSet}(\text{MSet}(\mathcal{A}), x)$

si $k = 0$ **alors renvoyer** M

pour i **de** 1 **à** $k - 1$ **faire**

$p_i \leftarrow \text{Poiss}(\frac{1}{i} A(x^i))$

répéter p_i **fois**

$\alpha \leftarrow \text{GenB}(\mathcal{A}, x^i)$

ajouter i copies **de** α à M

$p_k \leftarrow \text{Poiss}_{\geq 1}(\frac{1}{k} A(x^k))$ *# au moins un élément pour l'indice k max*
répéter p_k **fois**

$\alpha \leftarrow \text{GenB}(\mathcal{A}, x^k)$

ajouter k copies **de** α à M

renvoyer M

Tirage de l'indice maximal

On connaît la distribution de l'indice maximal K du produit (dans la formule du multi-ensemble) dans le modèle de Boltzmann :

$$\mathbb{P}_x(K = k) = \frac{\prod_{i=1}^k e^{A(x^i)/i}}{M(x)}.$$

On choisit un réel aléatoire U dans $[0, 1]$ et on trouve k tel que :

$$\frac{\prod_{i=1}^{k-1} e^{A(x^i)/i}}{M(x)} < U \leq \frac{\prod_{i=1}^k e^{A(x^i)/i}}{M(x)}, \text{ i.e., } \sum_{i=k+1}^{\infty} \frac{1}{i} A(x^i) < \log \frac{1}{U} \leq \sum_{i=k}^{\infty} \frac{1}{i} A(x^i)$$

Algorithme :

IndiceMaxMSet(MSet(\mathcal{A}), x)

$U \leftarrow \text{random}()$, $V \leftarrow \log \frac{1}{U}$, $p \leftarrow \log M(x)$, $k \leftarrow 0$

tant que $V < p$ **faire**

$k \leftarrow k + 1$

$p \leftarrow p - A(x^k)/k$

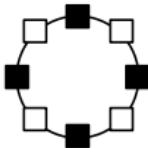
renvoyer k

Engendrer des cycles

Série génératrice de $\mathcal{C} = \text{Cyc}(\mathcal{A})$:

$$C(x) = \sum_{k \geq 1} \frac{\varphi(k)}{k} \log \frac{1}{1 - A(x^k)}.$$

On engendre les cycles sous formes de séquences répétant un motif.
Par exemple,



peut être engendré de différentes façons : $(\square \blacksquare)^4$, $(\blacksquare \square)^4$,
 $(\square \blacksquare \square \blacksquare)^2$, $(\blacksquare \square \blacksquare \square)^2$, $(\square \blacksquare \square \blacksquare \square \blacksquare \square \blacksquare \square)$ ou $(\blacksquare \square \blacksquare \square \blacksquare \square \blacksquare \square \blacksquare)$.

L'uniformité vient des distributions utilisées pour choisir le nombre de répétitions et la longueur du motif.

Engendrer des cycles - suite

GenB(Cyc(\mathcal{A}), x)

$k \leftarrow \text{Replic}(x)$ *# nombre de répétitions du motif*

$j \leftarrow \text{Loga}(A(x^k))$

pour i **de** 1 **à** j **faire**

$w_i \leftarrow \text{GenB}(\mathcal{A}, x^k)$

ajouter k copies **de** α à M

$w \leftarrow (w_1, w_2, \dots, w_j)$

renvoyer un cycle composé de k copies de w

Nombre de répétitions k du motif par découpage d'intervalle :

$$\mathbb{P}(K = k) = \frac{\varphi(k)}{kC(x)} \log \frac{1}{1 - A(x^k)}$$

La longueur j du motif suit une loi logarithmique de paramètre $A(x^k)$:

$$\mathbb{P}(J = j) = \frac{A(x^k)^j}{j} \left(\log \frac{1}{(1 - A(x^k))} \right)^{-1}$$

Ensemble sans répétitions PSet

Pour $\text{PSet}(\mathcal{A})$, on ne peut pas utiliser des appels indépendants au générateur de \mathcal{A} et on veut éviter le rejet. On utilise la décomposition d'un multi-ensemble induite par l'identité de Vallée :

$$\text{MSet}(\mathcal{A}) \cong \text{PSet}(\mathcal{A}) \times \text{MSet}(\Delta_2 \mathcal{A}).$$

↑ ↑
éléments isolés paires d'éléments identiques

On obtient un générateur (surjectif) d'ensembles sans répétitions de \mathcal{A} à partir du générateur de multi-ensemble **GenB**($\text{MSet}(\mathcal{A}, x)$) :

```
GenB( $\text{PSet}(\mathcal{A}), x$ )
   $M \leftarrow \text{GenB}(\text{MSet}(\mathcal{A}), x)$ 
   $P \leftarrow \emptyset$ 
  pour  $\alpha$  dans  $M$  faire
    si la mulitpliéte de  $\alpha$  dans  $M$  est impaire alors
      ajouter  $\alpha$  à  $P$ 
    renvoyer  $P$ .
```

1 Contexte

2 Méthode Récursive

3 Interlude : créer automatiquement un générateur à partir d'une spécification

4 Méthode de Boltzmann

- Un mot sur les arbres de Bienaymé-Galton-Watson
- Le principe
- Algorithmes récursifs de génération (sans symétrie)
- Exemples de générateur "libres"
- Générateurs pour les constructions avec symétries
- Exemples et applications

5 Questions d'implémentation

Partitions d'entiers

Spécifications pour le partitions d'entiers (\mathcal{P}) et celles en sommants distincts \mathcal{Q} :

$$\mathcal{P} = \text{MSet}(\text{Seq}_{\geq 1}(\mathcal{Z})) \quad \text{et} \quad \mathcal{Q} = \text{PSet}(\text{Seq}_{\geq 1}(\mathcal{Z})).$$

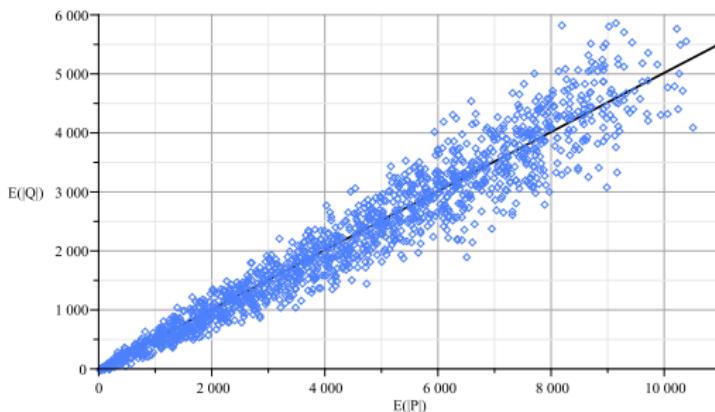
- On obtient facilement un générateur **GenB**(\mathcal{P}, x) pour les partitions d'entiers à partir de **GenB**(MSet, x).
- En tirant un réel u aléatoire dans $[0, 1]$ et en renvoyant $\ln(u)/\ln(x)$, on a une Geom(x) de une complexité arithmétique $O(1)$.
- Le coût d'un tirage d'une partition de taille n est donc linéaire en son nombre de sommants, c'est à dire $O(\sqrt{n})$ en moyenne.
- La distribution des tailles des partitions est concentrée, le générateur **GenB**(\mathcal{P}, x) permet de tirer des partitions dont la taille peut aller jusqu'à 10^{10} en quelques minutes.

Programme : en TP

Partitions d'entiers en sommants distincts

Le générateur $\text{GenB}(Q, x)$ pour les partitions en sommants distincts, est obtenu par extraction à partir du précédent.

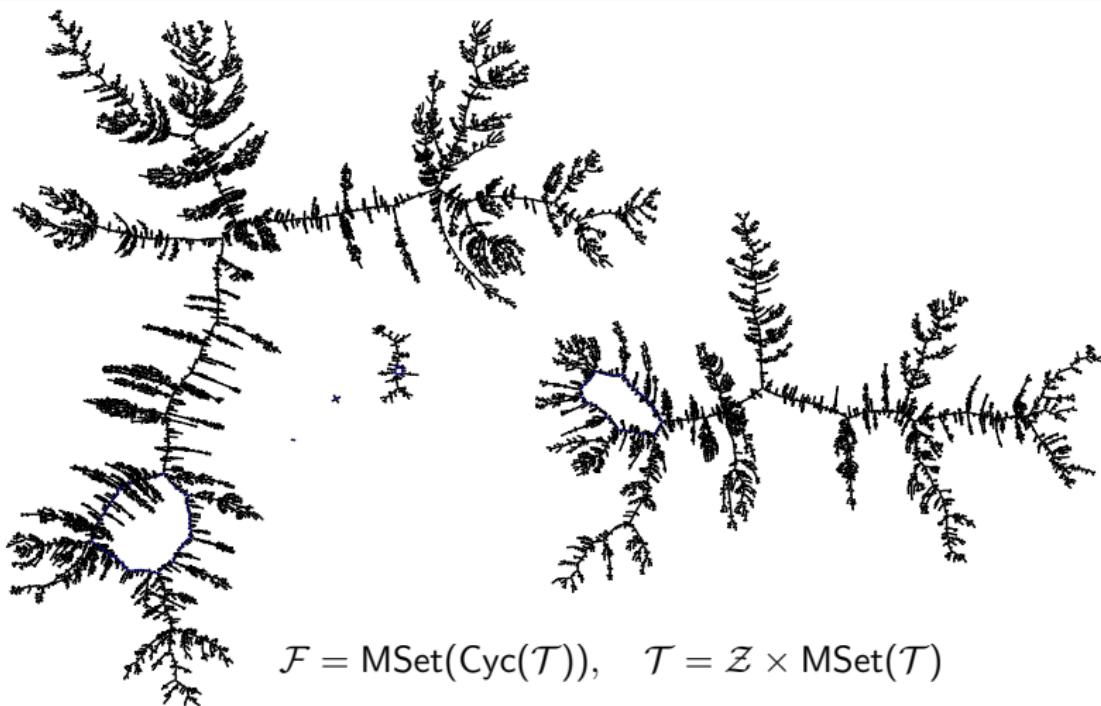
Le surcoût est proportionnel à la taille de la partition. En pratique, on observe que le rapport entre la taille d'une partition en sommants distincts et celle de la partition correspondante est d'environ 1/2.



Taille des partitions en sommants distincts en fonction de la taille des partitions d'entier correspondantes (en noir : rapport des espérances de tailles)

Programme : en TP

Graphes fonctionnels



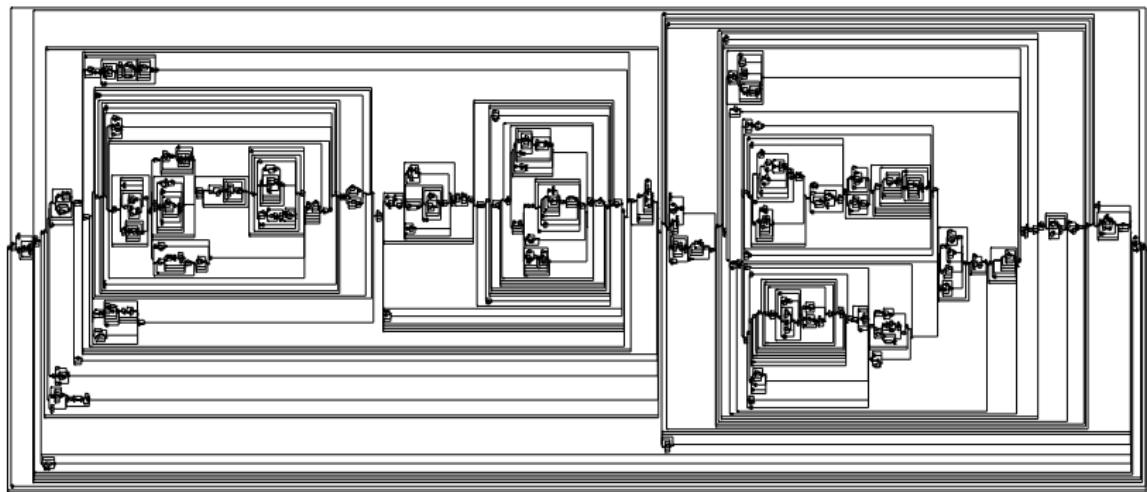
$$\mathcal{F} = \text{MSet}(\text{Cyc}(\mathcal{T})), \quad \mathcal{T} = \mathcal{Z} \times \text{MSet}(\mathcal{T})$$

Avec comme paramètre $x = 0.33831$ (assez proche de la singularité $\rho_{\mathcal{F}} = \rho_{\mathcal{T}}$), pour viser une taille de 20 000 noeuds, les graphes engendrés sont de taille : 9353, _, 4065, 18328, 12667, 3757, 4322, 653, 3844, 8809,

Programme : en TP

Circuits série-parallèle

$$\mathcal{SP} = \mathcal{Z} + \mathcal{S} + \mathcal{P}, \quad \mathcal{S} = \text{Seq}_{\geq 2}(\mathcal{Z} + \mathcal{P}), \quad \mathcal{P} = \text{MSet}_{\geq 2}(\mathcal{Z} + \mathcal{S})$$



Programme : en TP (circuits étiquetés)

Liste non exhaustive de travaux connexes

Applications

- Automates accessibles déterministes, *F. Bassino, J. David, C. Nicaud*
- Graphes Planaires, *E. Fusy*
- Partition planes, *O. Bodini, E. Fusy, C. Pivoteau*
- Réseaux Apolloniens, *A. Darrasse, M. Soria*
- Structures d'ARN, *Y. Ponty*
- Test de logiciel, *J. Oudinet*
- Diamants croissants, *O. Bodini, M. Dien, X. Fontaine, A. Genitrini, H-K. Hwang*
- ...

Extensions

- Opérateur de pointage non biaisé pour les structures non étiquetées, *M. Bodirsky, E. Fusy, M. Kang, S. Vigerske*
 - Structures colorées, *O. Bodini, A. Jacquot*
 - Produit ordonné, shuffle, produit de Hadamard, *A. Darrasse, K. Panagiotou, O. Roussel, M. Soria + A. Sportiello*
 - Générateur de lambda termes, *O. Bodini, D. Gardy, A. Jacquot*
 - Générateurs de Boltzmann multivariés, *O. Bodini, Y. Ponty*.
 - Arbres de Motzkin en temps linéaire, *A. Bacher, O. Bodini, A. Jacquot*
 - Chemins dirigés inhomogènes, *F. Bassino, A. Sportiello*
 - Optimisation convexe pour le paramétrage, *M. Bendkowski, O. Bodini, S. Dovgal*
 - Arbres enrichis en temps linéaire, *K. Panagiotou, L. Ramzews, B. Stufler*
 - Structures irréductibles et context-free en temps linéaire, *A. Sportiello*
 - ...
-
- Propriétés des graphes aléatoires via les gén. de Boltzmann, *K. Panagiotou, A. Weißl*

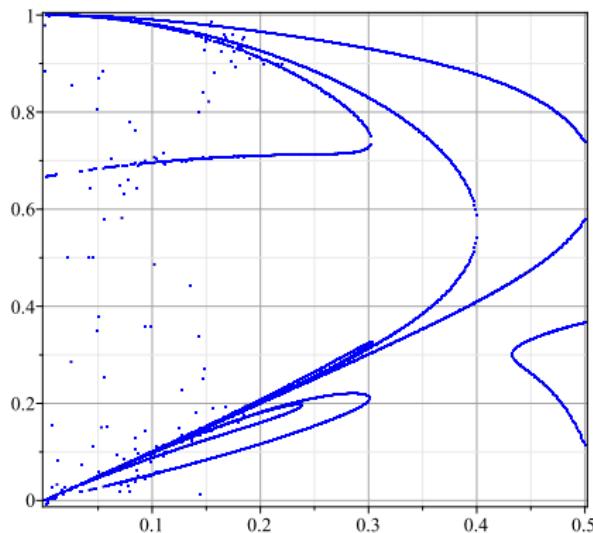
- 1 Contexte
- 2 Méthode Récursive
- 3 Interlude : créer automatiquement un générateur à partir d'une spécification
- 4 Méthode de Boltzmann
- 5 Questions d'implémentation

- 1 Contexte
- 2 Méthode Récursive
- 3 Interlude : créer automatiquement un générateur à partir d'une spécification
- 4 Méthode de Boltzmann
- 5 Questions d'implémentation
 - Comment obtenir un oracle ?
 - Génération en taille approchée/fixée

Évaluation numérique de la solution d'un système

$$Y_1 = x + \frac{x Y_4 Y_3 Y_1}{1 - Y_4}, \quad Y_2 = x + x Y_3 Y_2 \left(Y_2 + \frac{1}{1 - Y_2 Y_4 x^2} \right)$$

$$Y_3 = 2x + \frac{x^2}{1 - (Y_2 x + 2x) Y_3}, \quad Y_4 = x \left(\frac{Y_3 Y_4^2}{1 - Y_1} + 2x \right)$$



Solutions réelles positives pour $x = 0.3$:

$\{Y_1 = 0.2049, Y_2 = 0.6064, Y_3 = 0.9508, Y_4 = 2.594\}$,
 $\{Y_1 = 0.2149, Y_2 = 0.5310, Y_3 = 0.8585, Y_4 = 2.856\}$,
 $\{Y_1 = 0.3011, Y_2 = 68.87, Y_3 = 0.05480, Y_4 = 0.1808\}$,
 $\{Y_1 = 0.3130, Y_2 = 59.80, Y_3 = 0.5910, Y_4 = 0.1892\}$,
 $\{Y_1 = 0.3194, Y_2 = 0.4801, Y_3 = 0.8400, Y_4 = 0.1939\}$,
 $\{Y_1 = 0.3241, Y_2 = 0.5772, Y_3 = 1.008, Y_4 = 0.1974\}$,
 $\{Y_1 = 0.7273, Y_2 = 0.5834, Y_3 = 0.9999, Y_4 = 0.6620\}$,
 $\{Y_1 = 0.7607, Y_2 = 0.4867, Y_3 = 0.8421, Y_4 = 0.7057\}$,
 $\{Y_1 = 0.8298, Y_2 = 15.46, Y_3 = 0.5525, Y_4 = 0.7939\}$,
 $\{Y_1 = 0.9306, Y_2 = 18.17, Y_3 = 0.2027, Y_4 = 0.9177\}$

← Solutions réelles positives pour Y_1 .

Une solution

*Algorithms for combinatorial structures :
Well-founded systems and Newton iterations,
C. Pivoteau, B. Salvy, M. Soria, 2012*

Calcul par itération

Oracle de Boltzmann :
itération **numérique** qui **converge**
vers l'unique solution "pertinente"

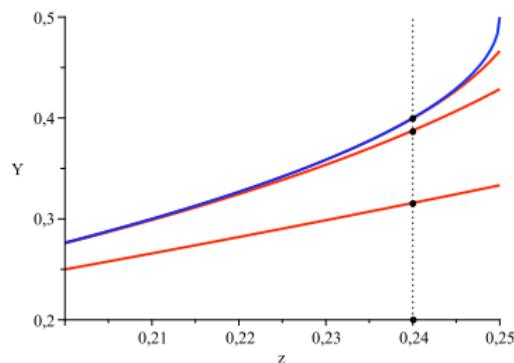
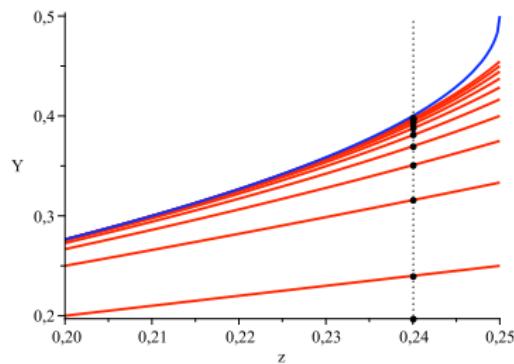


convergence de l'itération sur
les séries de **dénombrement**



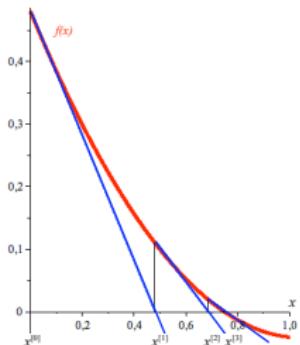
convergence de l'itération pour
les systèmes d'équations **combinatoires**

Efficacité : itération de Newton



Arbres binaires : $B(z) = z + B(z)^2$

L'itération de Newton



résoudre $f(x) = 0$:

$$x^{[n+1]} = x^{[n]} - \frac{f(x^{[n]})}{f'(x^{[n]})}$$

résoudre $y = f(y)$:

$$y^{[n+1]} = y^{[n]} + \frac{1}{1 - f'(y^{[n]})} (f(y^{[n]}) - y^{[n]})$$

$$\mathcal{Y}^{[n+1]} = \mathcal{Y}^{[n]} + \left(\mathbf{1} - \frac{\partial \mathcal{H}}{\partial \mathcal{Y}}(\mathcal{Z}, \mathcal{Y}^{[n]}) \right)^{-1} (\mathcal{H}(\mathcal{Z}, \mathcal{Y}^{[n]}) - \mathcal{Y}^{[n]}), \quad \mathcal{Y}^{[0]} = \mathbf{0}$$

$$\mathcal{Y}^{[0]} = \emptyset$$

$$T = \mathcal{Z} \times \text{SEQ}(T)$$

$$\mathcal{Y}^{[1]} = \boxed{\bullet} \boxed{\bullet} \boxed{\bullet} \dots$$

$$\mathcal{Y}^{[2]} = \boxed{\bullet} \boxed{\bullet} \boxed{\bullet} \dots \boxed{\bullet} \boxed{\bullet} \boxed{\bullet} \dots \boxed{\bullet} \boxed{\bullet} \dots$$

Démo en Maple

1 Contexte

2 Méthode Récursive

3 Interlude : créer automatiquement un générateur à partir d'une spécification

4 Méthode de Boltzmann

5 Questions d'implémentation

- Comment obtenir un oracle ?
- Génération en taille approchée/fixée

Choisir le paramètre pour GenB(\mathcal{C}, x)

$$\mathbb{E}_x(N) = x \frac{C'(x)}{C(x)}, \quad \mathbb{E}_x(N^2) = x^2 \frac{C''(x)}{C(x)} + x \frac{C'(x)}{C(x)}, \quad \sigma_x = \mathbb{E}_x(N^2) - \mathbb{E}_x(N)^2$$

• Distribution concentrée

$$\lim_{x \rightarrow \rho^-} \mathbb{E}_x(N) = +\infty \quad \text{et} \quad \lim_{x \rightarrow \rho^-} \sigma_x / \mathbb{E}_x(N) = 0$$

Exemples : partitions d'entiers, partitions d'ensembles, ...

• Distribution “plate”

$$[z^n]C(z) \sim \frac{a_0}{\Gamma(\alpha)} \rho^{-n} n^{\alpha-1}, \quad (n \rightarrow \infty), \quad -\alpha < 0$$

Exemples : surjections, mots sans *long runs*, permutations, ...

Paramètre : $x_n \in [0, \rho[, \text{ solution de } n = x \frac{C'(x)}{C(x)}$

Complexité en moyenne : $O(n)$ en taille approchée et $O(n^2)$ en taille fixée

- **Distribution piquée**

Exemples : familles simples d'arbres (exposant singulier $-\alpha = 1/2$)

Complexité : $O(n^2)$ en taille approchée...

Pour revenir à une complexité moyenne en $O(n)$:

- pointage :
 - ▷ l'exposant singulier diminue, on revient dans le cas d'une distribution plate
 - ▷ la taille de la spécification augmente
- générateur singulier (cas Bienaymé-Galton-Watson critique)
 - ▷ rejet anticipé
 - ▷ nombre fixe de valeurs numériques nécessaires
 - ▷ séquences supercritiques : $O(n)$ en taille fixe

Générateurs :

- USAIN BOLTZ en Python : générateurs de Boltzmann

<https://gitlab.com/ParComb/usain-boltz>

*Automatic compile-time synthesis of entropy-optimal Boltzmann samplers,
M. Dien, M. Pépin, 2022*

- Paganini en Haskell : générateurs de Boltzmann paramétrés

<https://github.com/maciej-bendkowski/boltzmann-brain.git>

*Automatic compile-time synthesis of entropy-optimal Boltzmann samplers,
M. Bendkowski, 2022*

Oracle/paramètre :

- NewtonGF en Maple : séries, oracle, rayon, choix du paramètre

*Algorithms for combinatorial structures : Well-founded systems and Newton
iterations, C. Pivoteau, B. Salvy, M. Soria, 2012*

<http://perso.ens-lyon.fr/bruno.salvy/software/the-newtongf-package/>

- Paganini en Python : oracle, rayon, choix du paramètre

*Tuning as convex optimisation : a polynomial tuner for multi-parametric
combinatorial samplers. M. Bendkowski, O. Bodini, S. Dovgal, 2022*

Séance de travaux pratiques

Au choix, programmer un générateur de...

- ... **partitions d'ensemble**
 - ▷ étiqueté, distribution concentrée. **facile** ↗
- ... **circuits série-parallèle**
 - ▷ étiqueté, distribution piquée (gen. singulier). **moyen** ↗
- ... **partitions d'entiers, partitions en sommants distincts**
 - ▷ non étiqueté, distribution concentrée. **moyen** ↗
- ... **graphes fonctionnels**
 - ▷ non étiqueté, distribution "plate". **avancé** ↗
- .. **vos objets préférés**
 - ▷ à condition d'avoir une spécification combinatoire.

Langage : notebooks Jupyter (Python) fournis, mais vous pouvez utiliser le langage de votre choix.

tous les notebooks ↗

Fin de la deuxième partie

Principe de base, méthode récursive

- A calculus for the random generation of labelled combinatorial structures. *P. Flajolet, P. Zimmermann, B. Van Cutsem.* TCS, 1994
- Random generation of unlabelled structures. Uniform random generation for the powerset construction. *P. Zimmermann (summary by E. Muray).* Algorithms Seminar, 1993-1994
- A Calculus of Random Generation : Unlabelled Structures. *P. Flajolet, P. Zimmermann, B. Van Cutsem.* draft, 1997

Principe de base, méthode de Boltzmann

- Boltzmann samplers for the random generation of combinatorial structures. *P. Duchon, P. Flajolet, G. Louchard, and G. Schaeffer.* CPC, 2004
- Boltzmann sampling of unlabelled structures. *P. Flajolet, É. Fusy, and C. Pivoteau.*, Analco 2007

Compléments

- mise en œuvre en TP
- de nombreuses applications existantes
- des extensions et améliorations récentes

This image was created with the assistance of DALL-E 2