Mémoire de Master d'Informatique.
Génération aléatoire et modèles de Boltzmann.

Carine Pivoteau

17 août 2009

Table des matières

Introduction

Les premiers générateurs aléatoires de structures combinatoires avaient pour but de rendre possibles des recherches de propriètés par échantillonage portant sur des analyses statistiques de classes combinatoires qu'il était impossible d'étudier de façon exhaustive. Nijenhuis et Wilf [6] ont alors proposé une méthode récursive pour concevoir des générateurs aléatoires uniformes en taille fixée pour différentes classes combinatoires telles que des partitions, des permutations ou encore des arbres. Cette méthode a, par la suite, été généralisée et systématisée pour un large ensemble de classes combinatoires dites "décomposables" par Flajolet, Zimmermann et Van Cutsem [4]. C'est d'ailleurs cette méthode qui est implantée par la bibliothèque MAPLE : Combstruct.

Depuis, les applications de la génération aléatoire de structures combinatoires se sont diversifiées et se situent désormais dans des domaines aussi variés que l'algorithmique expérimentale (simulations), la physique, la bio-informatique ou encore le génie logiciel. Les besoins et les contraintes sur ces générateurs ont alors évolué et l'on développe maintenant des générateurs non plus uniformes mais pondérés ou encore des générateurs en taille non fixée. C'est dans cette seconde catégorie que se situent les générateurs de Boltzmann. L'idée principale sur laquelle ils reposent est qu'en relâchant un peu la contrainte de taille sur les objets générés, on arrive à gagner en complexité.

Alain Denise et son équipe de bio-informatique du LRI utilisent, par exemple, des générateurs aléatoires de structures secondaires d'ARN pour faire de l'analyse statistique de données génomiques (voir [7]). Étant donné la taille, souvent grande, des objets à engendrer, on recherche, si possible, des générateurs de complexité linéaire. On peut alors tout à fait tolérer de légers écarts de taille sur les objets générés. Nous verrons que les générateurs de Boltzmann, sur lesquels porte le travail présenté ici, sont particulièrement adaptés à ces besoins.

Ces générateurs trouvent également des applications pour le test de logiciel. En effet, les graphes de contrôle de programmes se formalisent comme des structures combinatoires décomposables dont les chemins (i.e., les exécutions possibles du programme) peuvent être générés aléatoirement, ce qui permet d'effectuer des tests statistiques (voir les travaux de S.-D. Gouraud [5]). Les chemins générés pour ces tests doivent avoir une longueur bornée mais non nécessairement fixe, ce qui est aisément réalisable avec un générateur de Boltzmann, comme le soulignent A. Denise, M.-C. Gaudel et S.-D. Gouraud [1].

Duchon, Flajolet, Louchard et Schaeffer [2] définissent précisément le modèle de Boltzmann. Selon ce modèle, plutôt que d'engendrer des objets avec une taille fixée, ont leur attribue un poids proportionnel à l'exponentielle de leur taille. De la sorte, on génère les objets de façon uniforme pour une taille donnée. La richesse de cette idée provient de ce que, pour les classes étiquetées spécifiables, il est possible de construire, de manière systématique, des générateurs aléatoires de complexité, le plus souvent, linéaire. Une panoplie d'algorithmes de génération aléatoire génériques a déjà été développée dans le cas étiqueté.

Le but de ce stage est de développer les éléments d'une théorie générale applicable aux classes combinatoires non étiquetées. Il s'agit notamment de prendre en compte les constructions d'Ensemble (PSet), de Multi-ensemble (MSet) et de Cycle (Cyc). Nous allons pour cela proposer des algorithmes génériques pour ces constructions ainsi que des générateurs spécifiques pour un certain nombre de classes combinatoires représentatives. Nous pourrons ainsi observer leur comportement en termes de distribution des tailles d'objets obtenus, et analyser leurs performances en termes de complexité.

Ce mémoire s'appuie sur une grande variété d'exemples. L'objectif général est de fournir, en même temps que des algorithmes génériques de générateurs pour les trois constructions mentionnées précédemment, plusieurs exemples de générateurs effectivement implantés, accompagnés d'optimisations portant sur les réglages des paramètres de contrôle. Nous fournirons également des données sur les performances résultantes de ces générateurs.

Plan. Un premier chapitre introductif présente trois générateurs d'arbres binaires, afin d'illustrer les différentes techniques de génération aléatoire étudiées précédemment. Les deux chapitres suivants traitent de la construction de multi-ensemble avec des générateurs de partitions d'entier et de structures arborescentes. On propose différents générateurs pour les partitions afin de se familiariser avec le modèle et d'illustrer, avec un premier exemple simple, la construction d'un générateur de multi-ensembles. Les générateurs d'arbres permettent ensuite de se pencher sur des classes récursives, plus complexes et d'aborder les techniques de réglage du paramètre de contrôle. Vient ensuite un chapitre sur les ensembles sans répétitions et enfin celui sur les cycles. Pour chaque classe combinatoire étudiée, on propose un générateur (son algorithme détaillé). On donne également les éléments de calculs (de singularités et de valeur de séries génératrices) qui permettent de régler le paramètre du générateur dans le but d'obtenir des objets de grande taille. On donne enfin des résultats en termes de distribution qui permettent d'apprécier le comportement du générateur sur un grand nombre de tirages aléatoires. Ce travail s'achève sur un récapitulatif des résultats théoriques et expérimentaux obtenus.

On donne, en annexe, les preuves de validité des algorithmes génériques pour les multi-ensembles et les cycles.

Enfin, un document annexe regroupant le code[1], produit pour cette étude, vient compléter ce mémoire.

[1] en Maple.

Chapitre 1

Génération aléatoire et modèle de Boltzmann

Classiquement, on s'intéresse à la génération aléatoire en taille fixée. On a tout d'abord utilisé des méthodes ad-hoc pour générer des objets de taille n fixée, suivant une distribution uniforme. Parmi celles-ci se trouve la méthode bijective, qui comme son nom l'indique, repose sur les bijections existant entre différentes familles d'objets combinatoires. Par la suite, on a utilisé les spécifications des classes combinatoires[1] pour automatiser la génération aléatoire. Cette méthode, dite méthode récursive [4], fournit des générateurs de complexité $O(n \log n)$ à $O(n^2)$ et nécessite un pré-traitement coûteux et beaucoup de mémoire.

La méthode de Boltzmann [2], quant à elle, consiste à engendrer des objets aléatoires en taille non fixée, avec une distribution répartie sur l'ensemble de la classe combinatoire, et uniforme sur chaque ensemble d'objets de même taille. Elle présente l'avantage de fournir des algorithmes de complexité $O(n)$ le plus souvent et d'être très peu coûteuse en mémoire.

Dans ce chapitre nous allons vous présenter ces trois méthodes en nous basant sur un exemple simple : les arbres binaires planaires.

1.1 La méthode bijective

Soient deux classes combinatoires $\mathcal{A}$ et $\mathcal{B}$ et φ une bijection entre ces deux classes. Supposons que l'on dispose d'un générateur aléatoire genA(n) pour $\mathcal{A}$; la méthode bijective consiste alors à utiliser φ pour transformer genA(n) en un générateur genB(n) pour $\mathcal{B}$.

Pour les arbres binaires, on utilise la bijection avec les mots de Dyck (ou mots de parenthèses). En effet, il est possible de fabriquer un générateur en utilisant le fait que l'on peut obtenir un mot de Dyck de longueur n à partir d'un mot binaire quelconque de longueur $n+1$ (par conjugaison). On utilise, en quelque sorte, une méthode surjective, pour passer des mots quelconques de longueur $n+1$, aux mots de Dyck de longueur n.

Nous proposons ici un générateur aléatoire qui n'utilise pas cette bijection-ci et que l'on pourrait également qualifier de surjectif, mais qui nous semble plus pédagogique. Il est basé sur la bijection entre deux familles d'arbres binaires. Il s'agit de l'algorithme de Rémy [8].

Nous considérons des arbres binaires planaires non étiquetés à n nœud internes[2]. On note B_n le nombre de tels arbres. On utilise également des arbres du même type avec des nœuds externes numérotés pour établir une relation entre les arbres à n nœuds internes et les arbres à $n-1$ nœuds internes. On note B'_n le nombre d'arbres binaires à nœuds externes numérotés.

On constate que pour obtenir un arbre à nœuds externes numérotés de taille n à partir d'un arbre à nœuds externes numerotés de taille $n-1$, il faut insérer un nouveau nœud interne au niveau d'un nœud déjà existant (interne ou externe). Cela peut se faire de deux façons différentes (à droite ou à gauche). Il y a donc,

[1] voir section 1.2.

[2] On remarque qu'un arbre à n nœuds internes est également un arbre à $n+1$ nœuds externes, i.e., un abre à $2n+1$ nœuds en tout.

en tout, $2(2n-1)$ façons de faire. On a alors la relation suivante :

$$B'_n = 2(2n-1)B'_{n-1}$$

De plus, il y a $(n+1)!$ façons de numéroter les nœuds externes d'un arbre binaire, donc :

$$B'_n = (n+1)!B_n$$

On a donc la relation suivante :

$$(n+1)B_n = 2(2n-1)B_{n-1}$$

On peut lire la relation précédente comme une bijection entre les arbres à n nœuds internes avec un nœud externe marqué (on les note $\mathcal{B}^+$) et les arbres à $n-1$ nœuds internes avec un nœud (interne ou externe) pointé par un marqueur $\in \{g, d\}$, représentant la gauche ou la droite (on les note $\mathcal{B}^\#$).

On passe alors d'un arbre de $\mathcal{B}^\#$ à un arbre de $\mathcal{B}^+$ en remplaçant le nœud marqué x par un nœud dont le fils droit (resp. gauche) est une feuille et le fils gauche (resp. droit) est x si la marque est d (resp. g). De la même manière, on peut passer d'un arbre de $\mathcal{B}^+$ à un arbre de $\mathcal{B}^\#$. On note φ cette bijection.

On peut donc aisément construire un générateur aléatoire récursif d'arbres de taille n qui consiste à engendrer un arbre de taille $n-1$ avec un nœud externe marqué et à le transformer, par φ, en un arbre de taille n dont on supprimera la marque.

On implante donc la bijection φ (procédure 1.1) et l'algorithme de génération (algorithme 1.1).

Procédure 1.1 $\varphi(A, i, m)$

si m=0 **alors** { la marque est à gauche}
  $N \leftarrow \blacksquare\langle\ \square\ ,$ Nœud$(A, i)\ \rangle$;
sinon { la marque est à droite}
  $N \leftarrow \blacksquare\langle$ Nœud$(A, i)\ ,\ \square\ \rangle$;
fin si
Retourner Remplacer_nœud(A, i, N) ;

où Nœud(A, i) renvoie le i-ème nœud de A dans l'ordre infixe et Remplacer_nœud(A, i, N) renvoie A dans lequel on a remplacé le i-ème nœud par N.

1.2 La méthode récursive

Comme beaucoup d'autres classes combinatoires, les arbres binaires forment une classe décomposable, i.e., ils admettent une spécification, à partir de $\{\varepsilon, \mathcal{Z}\}$ (respectivement l'élément de taille 0 et l'atome de taille 1) sur l'ensemble des constructions suivantes [4] :

{+, . , séquence, ensemble, multi-ensemble, cycle}

Algorithme 1.1 genB_bijectif(n) : générateur bijectif d'arbres binaires

si $n = 1$ **alors**
  Retourner $\blacksquare\langle\ \square\ ,\ \square\ \rangle$;
sinon
  $A \leftarrow$ genB_bijectif$(n-1)$;
  $i \leftarrow$ random$(1, 2n-1)$;
  $m \leftarrow$ random$(0, 1)$;
  Retourner Retirer_marque$(\varphi(A, i, m))$;
fin si

En effet, un arbre binaire est, soit une feuille, soit le produit de deux arbres binaires. La spécification pour les arbres binaires est alors :

$$\mathcal{B} = \mathcal{Z} + \mathcal{B} \times \mathcal{B} \tag{1.1}$$

Á cette spécification, on peut associer la série génératrice définie par la relation suivante[3] :

$$B(x) = \sum_{b \in \mathcal{B}} x^{|b|}$$

Ce qui peut également s'écrire sous la forme[4] :

$$B(z) = \sum_{n=0}^{\infty} B_n z^n$$

On a donc, d'après [3], et à partir de l'équation (1.1),

$$B(z) = z + B(z)^2$$

Cette série nous permet de déterminer les valeurs des coefficients B_n qui seront utilisés par l'algorithme de génération.

L'idée de base pour concevoir un générateur aléatoire d'arbres binaires en taille fixée, est d'utiliser un processus d'éclatement respectant les probabilités qui permettent d'obtenir une distribution uniforme.

Supposons que l'on souhaite engendrer un arbre de taille n. Si $n = 1$, d'après l'équation (1.1), la seule possibilité est de renvoyer un arbre composé d'une unique feuille. Si $n \geq 2$, la seule possibilité est de renvoyer un arbre composé de deux sous-arbres dont la somme des tailles vaut n. La probabilité que notre arbre de taille n aie une composante de taille k et une composante de taille $n-k$ est alors :

$$\binom{n}{k} \frac{B_k B_{n-k}}{B_n^{(2)}}$$

[3] $|b|$ désigne la taille de l'arbre b.
[4] B_n est le nombre d'objets de taille n dans $\mathcal{B}$.

où

$$B_n^{(2)} = [z^n]B(z)^2 = \sum_{o \le k \le n} B_k B_{n-k}$$

Pour tirer k, on peut donc utiliser la procédure 1.2.

Procédure 1.2 Tirer_k($\boldsymbol{n}$)

$U \leftarrow \text{random}(0,1)$;
$K \leftarrow 0$;
$S \leftarrow B_0 B_n / B_n^{(2)}$;
tant que $U \geq S$ **faire**
  $K \leftarrow K + 1$;
  $S \leftarrow S + B_k B_{n-k} / B_n^{(2)}$;
fin tant que
Retourner K ;

On obtient alors, aisément, le générateur d'arbres binaires décrit par l'algorithme 1.2.

Algorithme 1.2 genB_récursif(n) : générateur récursif d'arbres binaires

si $n = 1$ **alors**
  Retourner □
sinon
  $k \leftarrow$ Tirer_k(n) ;
  Retourner ■⟨ genB(k) , genB($n-k$) ⟩ ;
fin si

On constate que cet algorithme utilise les coefficients B_n et $B_n^{(2)}$, pré-calculés à partir de $B(z)$. Ce pré-traitement se fait en $O(n^2)$ en temps et en $O(n)$ en espace (voir [4]).

Cette technique est systématisée dans [4] et permet de fournir des générateurs pour toutes les classes décomposables. C'est celle qu'implante la bibliothèque Combstruct de MAPLE, dédiée à la combinatoire. Les algorithmes qu'elle produit on une complexité $O(n \log n)$ à $O(n^2)$.

1.3 Le modèle de Boltzmann

L'inconvénient de la méthode précédente est principalement le pré-traitement, lequel nécessite des calculs arithmétiques en multi-précision et le stockage des coefficients correspondants. De plus, on aimerait avoir des algorithmes de génération qui soient linéaires en temps afin de pouvoir engendrer des objets de grande taille. On va voir, avec les générateurs de Boltzmann, que c'est possible, à condition de relâcher légèrement la contrainte de taille sur les objets générés.

Au lieu de fabriquer un générateur genB(n) dont le paramètre n est la taille de l'arbre que l'on souhaite engendrer, on va maintenant concevoir un générateur $\Gamma B(x)$ dont le paramètre x sert à modifier l'espérance de la taille des arbres engendrés. En effet, le modèle de Boltzmann est défini comme suit :

Définition 1.1 *Dans le cas non étiqueté, le modèle de Boltzmann assigne à tout objet $b \in \mathcal{B}$ la probabilité suivante :*

$$\mathbb{P}_x(b) = \frac{x^{|b|}}{B(x)}$$

Un générateur de Boltzmann $\Gamma B(x)$ pour la classe $\mathcal{B}$ est un algorithme qui produit des objets de $\mathcal{B}$ suivant ce modèle.

Ainsi, deux arbres de même taille ont la même probabilité d'être tirés. D'après l'équation (1.1), la probabilité de tirer un objet de taille N avec un générateur qui respecte ce modèle est :

$$\mathbb{P}_x(N = n) = \frac{B_n x^n}{B(x)}$$

Cela nous permet de déterminer la distribution en taille des arbres générés par $\Gamma B(x)$ en fonction de la valeur de x.

Afin d'obtenir un générateur de Boltzmann pour les arbres binaires, on reprend l'idée du processus de branchement, mais en appliquant les probabilités qui correspondent au modèle. Donc pour engendrer un arbre, (paramétré par x), on renvoie $\square$ avec probabilité $x/B(x)$ ou $\blacksquare\langle\, \Gamma B(x), \Gamma B(x)\, \rangle$ avec probabilité $1 - x/B(x)$ (voir algorithme 1.3).

Algorithme 1.3 $\mathbf{\Gamma} B(x)$: générateur de Boltzmann d'arbres binaires

$b \leftarrow \text{Bern}(x/B(x))$;
si $b = 1$ **alors**
  Retourner $\square$
sinon
  Retourner $\blacksquare\langle\, \Gamma B(x), \Gamma B(x)\, \rangle$;
fin si

On constate que, contrairement à la méthode bijective, dans le cas du produit, il n'est pas nécessaire de déterminer les tailles des sous-arbres à l'avance, puisque la taille de l'objet à générer n'est pas fixée. Pour l'union, le choix se fait simplement grâce à une loi de Bernoulli de paramètre $x/B(x)$.

De plus, on n'utilise plus les coefficients des séries génératrices mais les *valeurs* des séries elles-mêmes. La complexité du prétraitement devient alors $O(1)$ en mémoire. L'algorithme n'utilise qu'un nombre fini de valeurs, on considère ici, que ce calcul, d'une autre nature, peut être confié à un oracle ; sa complexité n'est donc pas prise en compte dans le temps de génération.

TAB. 1.1 – Générateurs de Boltzmann

construction	générateur
cas non étiqueté	
$\mathcal{C} = \emptyset$	$\Gamma C(x) = \varepsilon$
$\mathcal{C} = \mathcal{Z}$	$\Gamma C(x) = z$
$\mathcal{C} = \mathcal{A} + \mathcal{B}$	$\Gamma C(x) = \left(\text{Bern}\left(\frac{A(x)}{A(x)+B(x)}\right) \longrightarrow \Gamma A(x) \mid \Gamma B(x)\right)$
$\mathcal{C} = \mathcal{A} \times \mathcal{B}$	$\Gamma C(x) = (\Gamma A(x); \Gamma B(x))$
$\mathcal{C} = \text{SEQ}(\mathcal{A})$	$\Gamma C(x) = (\text{Geom}(A(x)) \Longrightarrow \Gamma A(x))$
cas étiqueté	
$\mathcal{C} = \emptyset$	$\Gamma C(x) = \varepsilon$
$\mathcal{C} = \mathcal{Z}$	$\Gamma C(x) = z$
$\mathcal{C} = \mathcal{A} + \mathcal{B}$	$\Gamma C(x) = \left(\text{Bern}\left(\frac{\hat{A}(x)}{\hat{A}(x)+\hat{B}(x)}\right) \longrightarrow \Gamma A(x) \mid \Gamma B(x)\right)$
$\mathcal{C} = \mathcal{A} \times \mathcal{B}$	$\Gamma C(x) = (\Gamma A(x); \Gamma B(x))$
$\mathcal{C} = \text{SEQ}(\mathcal{A})$	$\Gamma C(x) = \left(\text{Geom}(\hat{A}(x)) \Longrightarrow \Gamma A(x)\right)$
$\mathcal{C} = \text{PSET}(\mathcal{A})$	$\Gamma C(x) = \left(\text{Pois}(\hat{A}(x)) \Longrightarrow \Gamma A(x)\right)$
$\mathcal{C} = \text{CYC}(\mathcal{A})$	$\Gamma C(x) = \left(\text{Loga}(\hat{A}(x)) \Longrightarrow \Gamma A(x)\right)$

Comme précédemment, cette méthode peut être systématisée. Le tableau 1.1 présente les résultats qu'ont obtenus Duchon, Flajolet, Louchard et Schaeffer dans [2] pour les classes d'objets étiquetés et pour l'union, le produit et la séquence dans le cas non étiqueté.

Le but de cette étude est de proposer une théorie générale dans le cas non étiqueté, notamment pour les constructions d'ensemble, de multi-ensemble et de cycle, afin de couvrir un large ensemble de classes combinatoires. Pour chaque construction, nous donnons un algorithme générique ainsi qu'une estimation du coût moyen de la génération en fonction de la taille moyenne n des objets que l'on engendre. Ce coût varie de $O(\sqrt{n})$ à $O(n)$. On peut ensuite, en utilisant une méthode de rejet, se placer dans un cadre de génération à taille n fixée (ou tout du moins, approchée), à condition de choisir le paramètre x du générateur de telle façon que l'espérance de la taille des objets engendrés :

$$\mathbb{E}_x(N) = x\frac{B'(x)}{B(x)}$$

soit égale à n.

Chapitre 2

Générateurs de Boltzmann pour les partitions d'entier : un premier pas vers les multi-ensembles

Ce chapitre à pour but de se familiariser avec les générateurs de Boltzmann pour des structures non étiquetées. Il nous permettra également d'introduire la méthode de génération que nous avons mise au point pour l'une des constructions principales de la combinatoire analytique : le multi-ensemble non étiqueté. Pour ce faire, nous allons présenter et étudier 3 générateurs :

- Un générateur de partitions d'entier dont le nombre de sommants est borné (section 2.2.1).
- Un premier générateur de partitions découlant directement de la représentation d'une partition comme un produit de séquences de sommants de taille donnée (section 2.3.1).
- Un second générateur basé sur la décomposition d'une partition générale en multi-ensemble d'entiers (section 2.3.2).

2.1 Les partitions d'entier

2.1.1 Définition

Une partition de l'entier n est un multi-ensemble de sommants $\{s_1, s_2, ..., s_k\}$ tel que $\sum_{i=1}^{k} s_i = n$. Les sommants ne sont pas nécessairement distincts et l'ordre des sommants n'est pas pris en compte.

Les partitions d'entier peuvent donc être vues comme des multi-ensembles d'entiers non nuls. Par ailleurs, les entiers strictements positifs, qui forment les sommants, sont des séquences non vides d'entiers, i.e., des produits d'atomes et de séquences d'atomes :

$$\mathcal{P} = \textsc{MSet}(\mathcal{I}),\ \ \mathcal{I} = \mathcal{Z} \times \textsc{Seq}(\mathcal{Z}) \tag{2.1}$$

On représente généralement ces partitions par des diagrammes de Ferrer. La figure 2.1 représente par exemple une partition aléatoire de taille 1000 générée par la librairie Combstruct de Maple. Chaque sommant est, ici, représenté par un rectangle horizontal.

2.1.2 Série génératrice

La décomposition (2.1) donne, par la théorie classique de combinatoire, la série génératrice suivante pour les partitions d'entiers :

$$P(z) = \exp\left(\sum_{k=1}^{\infty} \frac{1}{k} I(z^k)\right) = \exp\left(\sum_{k=1}^{\infty} \frac{z^k}{k(1-z^k)}\right) \tag{2.2}$$

Mais on peut également considérer les partitions quelconques comme des produits cartésiens infinis de séquences d'entiers de même taille :

$$\mathcal{P} = \textsc{Seq}(\mathcal{I}_{=1}) \times \textsc{Seq}(\mathcal{I}_{=2}) \times \textsc{Seq}(\mathcal{I}_{=3}) \times \dots$$

Ce qui nous donne une expression équivalente à l'équation (2.2) pour $P(z)$:

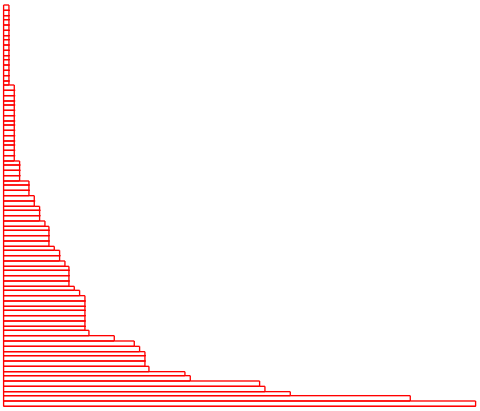


FIG. 2.1 – Partition aléatoire de taille 1000.

$$P(z) = \prod_{n=1}^{\infty} (1 - z^n)^{-I_n} = \prod_{n=1}^{\infty} (1 - z^n)^{-1} \tag{2.3}$$

2.2 Un cas simple : les partitions dont la taille des sommants est bornée

2.2.1 Générateur

Lorsque la taille des sommants est bornée par k, l'équation (2.3) devient :

$$P^{[k]}(z) = \prod_{n=1}^{k} \frac{1}{1 - z^n} \tag{2.4}$$

A partir de la série précédente, on construit donc le générateur $\Gamma P^{[k]}(x)$, où x est le paramètre de Boltzmann du générateur. Pour chaque taille de sommant jusqu'à k, on tire une loi géométrique (voir algorithme d'un générateur de Boltzmann pour une séquence, tableau 1.1), qui donne le nombre de sommants (éventuellemnt nul) de cette taille, dans la partition engendrée. On choisit une représentation compacte pour les partitions : des listes de couples [*taille de sommant, multiplicité du sommant*]. On a alors l'algorithme 2.1

Algorithme 2.1 $\Gamma P^{[k]}(x)$: générateur de partitions d'entier avec taille de sommant bornée

Retourner $\big(\ [1, \mathrm{Geom}(x)]\ ;\ \ldots\ ;\ [k-1, \mathrm{Geom}(x^{k-1})]\ ;\ [k, \mathrm{Geom}(x^k)]\ \big)$

Le tirage de la loi géometrique a une complexité $O(1)$. Ce générateur a donc une complexité $O(k)$.

On constate qu'engendrer un multi-ensemble d'entiers de cette façon ne pose aucune difficulté. Cela vient du fait que les objets qui composent le multi-ensemble sont très simples à générer. Pour des objets plus complexes, nous aurions besoin d'un générateur exhaustif, capable de générer tous les objets d'une taille donnée, ce qui pourrait s'avérer très gênant en termes de complexité. Nous verrons par la suite comment fabriquer des générateurs de Boltzmann pour les multi-ensembles qui ne présentent pas cette contrainte.

2.2.2 Mise en œuvre

On souhaite choisir x_0 de façon que la taille soit, avec une "bonne" probabilité, voisine d'une valeur fixée à l'avance (et grande) N. Pour ce faire, une heuristique est de choisir le paramètre x de la loi de Boltzmann tel que l'espérance de la taille d'une partition aléatoire soit voisine de N. On doit résoudre $N = x_0 \frac{P^{[k]\prime}(x_0)}{P^{[k]}(x_0)}$. On a donc $x_0 \sim 1 - \frac{k}{N}$. La valeur $x_N = 1 - \frac{k}{N}$ est, par conséquent, une bonne approximation de x_0. Avec les paramètres $x_0 = 9/10$ et $k = 10$, on obtient, sur 1000 tirages, une taille moyenne de partition d'environ 970.

A partir de la série génératrice bivariée $P_j^{[k]}(z, u)$ dont la variable u marque le nombre de sommants de taille j, on peut calculer l'esperance du nombre de sommants χ de taille j (fixée) dans une partition de taille N (fixée).

$$P_j^{[k]}(z, u) = \frac{1 - z^j}{1 - uz^j} \prod_{n=1}^{k} \frac{1}{1 - z^n}$$

$$\begin{aligned} E_{k,N}(\chi) &= \frac{\frac{\partial}{\partial u} P_j^{[k]}(z, u)\Big|_{u=1}}{P_j^{[k]}(z, u)\Big|_{u=1}} \\ &= \frac{\partial}{\partial u} \log\left(P_j^{[k]}(z, u)\right)\Big|_{u=1} \\ &= \frac{x_N^j}{1 - x_N^j} \\ &\approx \frac{N}{jk} \end{aligned}$$

La figure 2.2 présente une comparaison entre le profil type (établi grâce à l'expression de $E_{k,N}(\chi)$) d'une partition dont la taille de sommant est bornée à 30 et le profil moyen d'une partion aléatoire générée par $\Gamma P_{10}(1 - \frac{10}{1000})$:

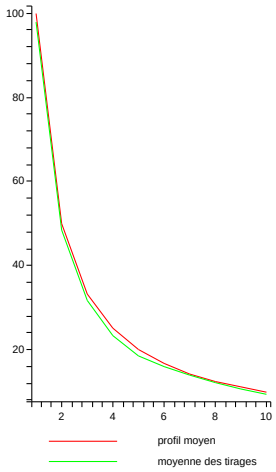


FIG. 2.2 – Profil type et profil moyen d'une partition aléatoire sur 1000 tirages de $\Gamma P^{[10]}(1 - \frac{10}{1000})$.

2.3 Cas général

2.3.1 Construction par produit

Générateur

Dans un premier temps, on cherche à construire un générateur basé sur l'équation (2.3). On veut simuler le tirage d'un produit infini. Il faut, pour cela, commencer par tirer la valeur k de la la taille du plus grand sommant. Ensuite on procède comme pour le générateur 2.1, en prenant soin d'avoir au moins un sommant de taille k.
On remarque que l'ensemble des partitions d'entiers peut être découpé en sous-ensembles de partitions suivant la taille k de leur plus grand sommant. On a donc :

$$P(z) = \sum_{k=1}^{\infty} P^{(k)}(z) = \sum_{k=1}^{\infty} \frac{z^k}{1-z^i} \prod_{j<k} \frac{1}{1-z^j}$$

où $P^{(k)}(z)$ est la série des partitions dont le plus grand sommant est k. On cherche alors K, pour U aléatoire dans $[0, 1]$, tel que :

$$\frac{\sum_{k=0}^{K} P^{(k)}(x)}{P(x)} \leq U \leq \frac{\sum_{k=0}^{K+1} P^{(k)}(x)}{P(x)}$$

Pour tirer K, on utilise donc l'algorithme implanté par la procédure 2.1 On obtient alors aisément le générateur $\Gamma P(x)$ (algorithme 2.2).

Procédure 2.1 PlusGrandSommant($\boldsymbol{x}$)

$U := \text{random}(0,1)$;
$p := \frac{P^{(1)}(x)}{P(x)}$;
$k := 1$;
tant que $U > p$ **faire**
  $k := k + 1$;
  $p = p + \frac{P^{(k)}(x)}{P(x)}$;
fin tant que
Retourner k

Algorithme 2.2 $\mathbf{\Gamma}P(x)$: générateur de partitions d'entier (produit)

$k \leftarrow$ PlusGrandSommant(x) ;
Retourner$\big($ $[1, \text{Geom}(x)]$; $\ldots$; $[k-1, \text{Geom}(x^{k-1})]$; $[k, \text{Geom}_{\geq 1}(x^k)]$ $\big)$

On peut calculer les $P^{(i)}$ de proche en proche, donc PlusGrandSommant(z) a une complexité $O(k)$ en nombre d'opérations arithmétiques sur les réels. On sait que le nombre de sommants d'une partition aléatoire de taille n est de l'ordre de $\sqrt{n}\log(n)$, en moyenne et qu'il existe une bijection échangeant le nombre de sommants et la taille du plus grand sommant. Donc K est, en probabilité, d'ordre $O(\sqrt{n}\log(n))$. Notre générateur a donc une complexité en moyenne $O(\sqrt{n}\log(n))$ en nombre d'opérations sur les réels.

Optimisation et mise en œuvre

L'algorithme de tirage du plus grand sommant donné précédemment préfigure le schéma général qui sera utilisé par la suite pour la génération de multi-ensembles. Néanmoins, pour les partitions, il est possible de déterminer la taille du plus grand sommant directement. On tire un réel U aléatoire dans $[0,1]$ et on cherche l'entier k tel que :

$$\frac{1}{P(x)}\prod_{i=1}^{k}\frac{1}{1-x^i} \leq U \leq \frac{1}{P(x)}\prod_{i=1}^{k+1}\frac{1}{1-x^i}$$

Ce qui amène :

$$\underbrace{\frac{x^k}{1-x}}_{A} + \frac{x^{2(k)}}{2(1-x)} + \cdots \geq \underbrace{\log\frac{1}{U}}_{V} \geq \frac{x^{k+1}}{1-x} + \frac{x^{2(k+1)}}{2(1-x)} + \ldots$$

On approche alors k par :

$$k \sim \frac{\log V + \log(1-x)}{\log x}$$

Et si $V \leq A + \frac{x^{2(k+1)}}{2(1-x)}$, on prend $k+1$.

Ensuite on choisit le paramètre x_0 du générateur de façon à avoir "de bonnes chances" d'obtenir des partitions de taille N. Sachant que l'espérance de la taille est donnée par $x\frac{P'(x)}{P(x)}$, on doit désormais résoudre : $x_0\frac{P'(x_0)}{P(x_0)} = N$. On a

$$x\frac{P'(x)}{P(x)} \sim_{x\to 1} \frac{\pi^2}{6(1-x)^2}$$

donc le réel x_N tel que $N = \frac{\pi^2}{6}\cdot\frac{1}{(1-x_N)^2}$ donne une bonne approximation de x_0. On choisit donc $x_N = 1 - \pi/\sqrt{6N}$

Ce générateur nous a ainsi permis de générer en MAPLE des partitions aléatoires de taille $\sim 10^{10}$ en quelques minutes. La figure 2.3 nous montre par exemple une partition de taille $\sim 10^6$ engendrée avec $\Gamma P(1 - \pi/\sqrt{6.10^6})$

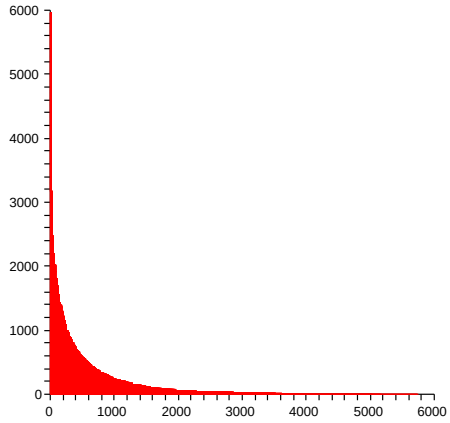


FIG. 2.3 – Partition de taille $\sim 10^6$

2.3.2 Construction par multi-ensemble

Générateur

Cette fois-ci, on va construire un générateur qui utilise l'équation (2.2). De cette façon, on n'a plus besoin de générer des entiers de façon exhaustive. Cette démarche sera généralisée par la suite et nous permettra de concevoir des générateurs de Boltzmann pour des multi-ensembles quelconques (voir chapitre 3). Si on réécrit cette équation, on a :

$$P(z) = \prod_{k=1}^{\infty} \exp\left(\frac{1}{k} I(z^k)\right)$$

Un générateur de Boltzmann pour les entiers non nuls se réduit au tirage d'une loi géométrique strictement positive. On dispose donc aisément du générateur $\Gamma I(x)$.

Á la série $I(z^k)$ correspond un "paquet" de k copies d'un même entier (voir Figure 2.4). Le générateur de Boltzmann associé engendre un entier I par $\Gamma I(x^k)$ et renvoie le produit de k copies de I.

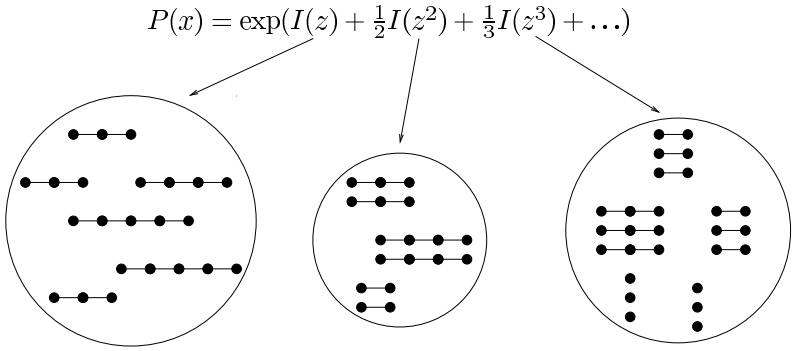


FIG. 2.4 – Construction d'une partition.

Pour générer les objets qui correspondent à $\exp\left(\frac{1}{k}I(x^k)\right)$, on utilise donc une loi de Poisson et on obtient le générateur $\Gamma S_k(x)$.

On note qu'on peut avoir de la même façon un générateur $\Gamma S_{\geq 1,k}(x)$, en utilisant une loi de Poisson non nulle.

Enfin, il faut déterminer la taille maximale de "paquet" k que l'on fait. Pour cela, on procède comme au paragraphe 2.3.1. On tire une variable U uniforme dans $[0, 1]$ et on cherche k tel que :

$$\frac{\exp\left(\sum_{i=1}^{k-1} \frac{1}{i} I(x^i)\right)}{P(x)} \leq U \leq \frac{\exp\left(\sum_{i=1}^{k} \frac{1}{i} I(x^i)\right)}{P(x)}$$

C'est à dire :

$$\sum_{i=k+1}^{\infty} \frac{x^i}{i(1-x^i)} \leq \log \frac{1}{U} \leq \sum_{i=k}^{\infty} \frac{x^i}{i(1-x^i)}$$

Pour tirer k, on utilise donc l'algorithme implanté par la procédure 2.2

Ce qui nous donne le générateur $\Gamma\tilde{P}(x)$ (algorithme 2.3).

Complexité de $\Gamma\tilde{P}(x)$

On dispose d'une procédure de tirage d'une loi géométrique de complexité $O(1)$. Grâce à la représentation compacte qu'on a choisie pour les partitions, la fonction $\Gamma I_k(x)$ a une complexité $O(1)$. Ainsi, la complexité de $\Gamma S_k(x)$ est en $O(P)$ où P est la valeur renvoyée par la loi de Poisson. La complexité globale du générateur est donc la somme des lois de Poisson tirées par les S_i.

Procédure 2.2 IndiceMaximalPaquet($\boldsymbol{x}$)

$U \leftarrow \text{random}(0,1)$;
$V \leftarrow \log \frac{1}{U}$;
$p \leftarrow \log P(x)$;
$k \leftarrow 0$;
tant que $V < p$ **faire**
  $k \leftarrow k+1$;
  $p \leftarrow p - \frac{x^k}{k(1-x^k)}$;
fin tant que
Retourner k ;

Algorithme 2.3 $\mathbf{\Gamma}\tilde{P}(x)$: générateur de partitions d'entier (exponentielle)

procédure $\mathbf{\Gamma}I_k(x)$
  $\mathrm{I} \leftarrow \Gamma I(x)$;
  Retourner ($\underbrace{\mathrm{I}\ ;\ \ldots\ ;\ \mathrm{I}}_{k \text{ fois}}$)
fin procédure

procédure $\mathbf{\Gamma}S_k(x)$
  $p \leftarrow \text{Pois}\left(\frac{x^k}{k(1-x^k)}\right)$
  Retourner ($\underbrace{\Gamma I_k(x^k)\ ;\ \ldots\ ;\ \Gamma I_k(x^k)}_{p \text{ fois}}$)
fin procédure

$k \leftarrow$ IndiceMaximalPaquet(x) ;
Retourner ($\Gamma S_1(x)\ ;\ \ldots\ ;\ \Gamma S_{k-1}(x)\ ;\ \Gamma S_{\geq 1,k}(x)$)

On note s_p la somme des lois de Poisson tirées par les S_i lors de la génération d'une partition p. Si tous les sommants tirés par $\Gamma I_k(x)$ pour générer p sont différents, alors s_p est égal au nombre de tailles distinctes de sommant apparaissant dans p. On a donc $s_p \leq \#$ *sommants distincts de* p. Or, [10] nous donne d_n, une estimation du nombre moyen de tailles différentes de sommant dans une partition de taille n :

$$d_n \sim \frac{\sqrt{6}}{\pi}\sqrt{n}$$

On en déduit que $\Gamma\tilde{P}(x)$ a un complexité en moyenne $O(\sqrt{n})$.

Calcul de l'oracle : $P(x)$

Le générateur précédent nécessite un oracle qui donne la valeur de $P(x)$. Dans un premier temps, nous avons calculé cette valeur de façon itérative grâce à l'équation (2.3). On calcule le produit jusqu'à ce que le terme multiplicateur soit inférieur à la précision recherchée. Dans les faits, x étant proche de 1, cette façon de calculer peut nécessiter un nombre important d'opérations. Afin de pallier à ce problème, on utilise la formule suivante :

$$P\left(\exp\left(-2\pi\tau\right)\right) = \sqrt{\tau}\,\exp\left(\frac{\pi}{12}\left(\frac{1}{\tau}-\tau\right)\right) P\left(\exp\left(-\frac{2\pi}{\tau}\right)\right)$$

Ce qui permet de se ramener à évaluer P en une valeur proche de 0. En pratique, le calcul de $P(x)$ se fait donc en temps constant.

2.4 Conclusion

Tout d'abord, l'exemple simple des partitions d'entier nous a permis d'illustrer la puissance, en termes d'efficacité et de simplicité, des générateurs de Boltzmann. Dans un deuxième temps , $\Gamma\tilde{P}(x)$ nous offre un premier exemple d'implantation de générateur de multi-ensemble. La technique employée ici sera en effet celle que nous utiliserons par la suite pour fabriquer des générateurs plus complexes.

Nous verrons, dans le chapitre suivant, comment implanter des générateurs de multi-ensembles récursifs ou avec des contraintes de cardinalité.

Chapitre 3

Générateurs de Boltzmann pour des structures arborescentes : le schéma général pour les multi-ensembles

Dans ce chapitre, nous allons nous intéresser en détail à la construction de multi-ensembles, au travers de plusieurs exemples de structures arborescentes, et notamment nous pencher sur la question des multi-ensembles avec contraintes de cardinalité. Nous allons également nous attarder quelques temps sur le réglage (par le choix du paramètre x et l'implémentation des oracles) des générateurs que nous proposons. Nous allons donc présenter trois générateurs :

- Un générateur d'arbres binaires non planaires. Celui-ci nous permettra d'introduire les multi-ensembles de cardinalité bornée (section 3.1.1).
- Un générateur d'arbres généraux non planaires enracinés (section 3.2.1).
- Un générateur de circuits série-parallèle. Ce dernier présente deux particularités intéressantes : il met en œuvre des multi-ensembles de cardinalité bornée et il a une structure relativement complexe qui exige quelques manipulations pour en effectuer les réglages (section 3.3).

3.1 Les arbres binaires non planaires enracinés

Ces arbres sont également connus sous le nom d'arbres d'Otter. Chaque sous-arbre d'un tel arbre est composé soit d'une feuille, soit de 2 sous-arbres. Comme ces arbres sont non planaires, on ne distingue pas le fils droit du fils gauche, i.e. chaque nœud interne est assimilé à un multi-ensemble de 2 sous-arbres. On a donc :

$$\mathcal{B} = \mathcal{Z} + \textsc{MSet}_2(\mathcal{B})$$

Cette décomposition se fait uniquement suivant les feuilles de l'arbre ; les nœuds internes ne sont pas pris en compte. Nous utiliserons la représentation suivante : □ pour une feuille et ■⟨ ⟩ pour un nœud interne. La figure 3.1 représente chacun des 6 arbres d'Otter différents de taille 6 (représentés sous forme de plongement arbitraires dans le plan).
Cette décomposition nous donne la série génératrice suivante :

$$B(z) = z + \frac{1}{2}B^2(z) + \frac{1}{2}B(z^2)$$

En particulier, $B(z)$ vérifie l'équation fonctionnelle :

$$B(z) = 1 - \sqrt{1 - 2z - B(z^2)}$$

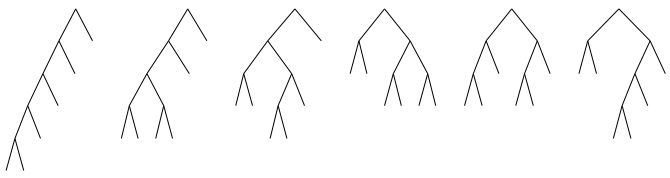


FIG. 3.1 – Les 6 arbres d'Otter de taille 6

3.1.1 Générateur

Pour une classe combinatoire $\mathcal{A}$ dont la série génératrice est $A(z)$, on définit $\mathcal{A}^{[k]}$ la classe dont les objets sont constitués par k copies d'un objet de $\mathcal{A}$ et dont la série génératrice est $A(z^k)$.

Proposition 3.1 *Etant donné un générateur de Boltzmann $\Gamma A(x)$ pour la classe $\mathcal{A}$, un générateur de Boltzmann $\Gamma A_k(x)$ pour la classe $\mathcal{A}^{[k]}$ s'obtient en faisant le produit de k copies d'un objet généré par $\Gamma A(x^k)$.*
Preuve :
La distribution de probabilité de la taille S d'un élément répété k fois dans $\mathcal{A}^{[k]}$ vérifie :

$$\mathbb{P}(S = n) = \frac{A_n x^{kn}}{A(x^k)} = \frac{A_n (x^k)^n}{A(x^k)}$$

C'est donc celle que l'on attend pour un générateur de Boltzmann. Par ailleurs, deux objets de même taille ont la même probabilité d'être tirés.

La proposition 3.1 nous permet alors de montrer que le générateur $\Gamma B(x)$ est un générateur de Boltzmann pour les arbres binaires non planaires :

Algorithme 3.1 $\mathbf{\Gamma} B(x)$: générateur d'arbres d'Otter

procédure $\mathbf{\Gamma} B_2(x)$
  $ot \leftarrow \Gamma B(x^2)$;
  Retourner $\blacksquare\langle$ ot , ot $\rangle$;
fin procédure

si Bern $\left(\frac{x}{B(x)}\right) = 1$ **alors**
  Retourner $\square$;
sinon si Bern $\left(\frac{B^2(x)}{B^2(x)+B(x^2)}\right) = 1$ **alors**
  Retourner $\blacksquare\langle\ \Gamma B(x)\ ,\ \Gamma B(x)\ \rangle$;
sinon
  Retourner $\Gamma B_2(x)$;
fin si

3.1.2 Mise en œuvre

On choisit de tirer des arbres en la singularité ρ de $B(x)$, car c'est la valeur pour laquelle la distribution de Boltzmann donne le plus d'arbres de grande taille, comme l'illustre la figure 3.2. Nous n'expliciterons pas le calcul de ρ, pour plus de détails, se reporter à [3] (chapitre VII.2.).

Notre générateur utilise un oracle pour les valeurs des $B(x^i)$. En utilisant l'équation (3.1), on voit que la singularité ρ est la valeur pour laquelle l'expression sous la racine carrée s'annule, ce qui implique $B(\rho) = 1$ et $B(\rho^2) = 1 - 2\rho$.

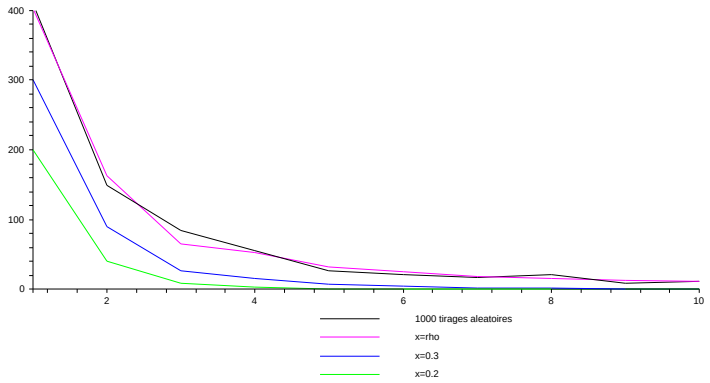


FIG. 3.2 – Distribution des tailles sous le modèle de Boltzmann pour les arbres d'Otter, pour différentes valeurs du paramètre x et distribution obtenue expérimentalement pour 1000 tirages aléatoires avec $x = \rho$.

Pour les autres valeurs de $B(x)$, on utilise le fait que $\lim_{x\to 0} B(x) = 0$ et on développe $B(x)$, pour k suffisamment grand :

$$B(x) = 1 - \sqrt{-2x - \sqrt{-2x^2 - \cdots - \sqrt{-2x^k}}}$$

On remarque que l'éspérance de la taille des arbres ainsi générés est infinie. Il est donc nécessaire de pouvoir limiter leur taille pendant la génération. Nous avons choisi, ici, par souci de simplicité, de rejeter les arbres dont le niveau de récursion (i.e. la longueur maximale de branche) est supérieur à 1000.

Pour 1000 tirages de $\Gamma B(\rho)$, on obtient une distribution en taille proche de celle espérée (voir figure 3.2). La figure 3.3 présente, par ailleurs, deux arbres d'Otter aléatoires, l'un généré par combstruct et l'autre par notre générateur.

3.2 Les arbres généraux non planaires enracinés

Cette fois-ci, le nombre de sous-arbres n'est pas limité. Les fils d'un nœud sont alors représentés par un multi-ensemble éventuellement vide de sous-arbres. On choisit désormais une décomposition suivant tous les nœuds de l'arbre (nœuds internes et feuilles) :

$$\mathcal{T} = \mathcal{Z} \times \mathrm{MSET}(\mathcal{T})$$

Cette décomposition nous donne la série génératrice suivante :

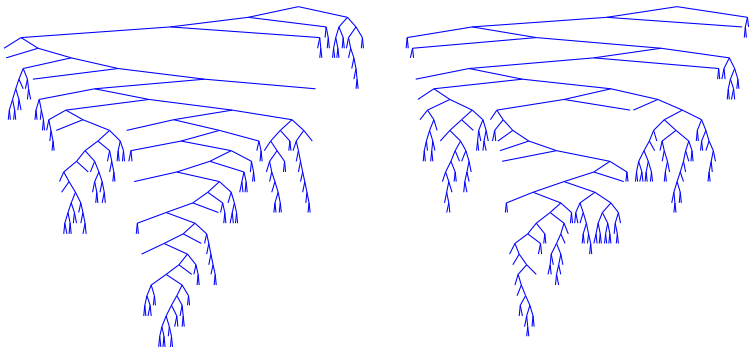


FIG. 3.3 – A gauche, un arbre d'Otter de taille 150 généré par combstruct, à droite un arbre aléatoire de taille 143, généré par $\Gamma B(\rho)$.

$$
\begin{aligned}
T(z) &= z \prod_{\gamma \in \mathcal{T}} \frac{1}{1 - z^{|\gamma|}} \\
&= z \prod_{n \geq 1} (1 - z^n)^{-T_n} \qquad (3.1) \\
&= z \exp\left(\sum_{k=1}^{\infty} \frac{1}{k} T(z^k) \right) \qquad (3.2)
\end{aligned}
$$

3.2.1 Générateur

Ce générateur possède à peu près la même structure que le second générateur de partitions d'entiers que nous avons déjà proposé. La différence majeure vient du fait que le générateur d'arbres est récursif.
On utilise l'équation (3.2). L'arbre que l'on génére est donc le produit d'un atome et d'un multi-ensemble d'arbres.

Comme pour les partitions, l'équation (3.2) indique comment décomposer le générateur $\Gamma T(x)$ en une séquence de générateurs $\Gamma S_k(x)$, tels que les objets générés dans $\Gamma S_k(x)$ sont regroupés par "paquets" de k copies identiques d'un objet de $\mathcal{T}$. On peut montrer que, presque sûrement, pour chaque tirage du multi-ensemble, il existe un indice k_0 tel que $\Gamma S_{k_0}(x)$ génère une ensemble non vide et tous les $\Gamma S_k(x)$ génèrent des ensembles vides pour $k > k_0$. Cet indice k_0 est appelé "indice maximal de paquet". Le générateur de multi-ensembles d'arbres que nous proposons fonctionne alors en deux étapes :

- d'abord, tirer l'indice maximal de paquet, k_0
- puis appeler les générateurs $\Gamma S_k(x)$ pour k allant de 1 à k_0, avec la condition que l'ensemble généré par $\Gamma S_{k_0}(x)$ soit non vide. Comme le suggère l'équation (3.2), chaque générateur $\Gamma S_k(x)$ va faire intervenir une loi de

Poisson[1] de paramètre $T(x^k)/k$, ce que traduit le facteur $\exp\big(T(z^k)/k\big)$ dans l'équation.

Pour tirer l'indice maximal de paquet k_0, on tire une variable U uniforme dans $[0,1]$ et on cherche k_0 tel que :

$$\frac{x\,\exp\left(\sum_{i=1}^{k_0-1}\frac{1}{i}T(x^i)\right)}{T(x)} \leq U \leq \frac{x\,\exp\left(\sum_{i=1}^{k_0}\frac{1}{i}T(x^i)\right)}{T(x)}$$

C'est à dire :

$$\sum_{i=k_0+1}^{\infty}\frac{1}{i}T(x^i) \leq \log\frac{1}{U} \leq \sum_{i=k_0}^{\infty}\frac{1}{i}T(x^i)$$

Pour tirer k_0, on utilise donc l'algorithme implanté par la procédure 3.1

Procédure 3.1 IndiceMaximalPaquet($\boldsymbol{x}$)

$U \leftarrow \text{random}(0,1)$
$V \leftarrow \log\frac{1}{U}$
$p \leftarrow \log\frac{T(x)}{x}$
$k \leftarrow 0$
tant que $V < p$ **faire**
  $k \leftarrow k+1$
  $p = p - T(x^k)/k$
fin tant que
Retourner k

Ainsi, on obtient le générateur $\Gamma T(x)$ pour les arbres généraux (algorithme 3.2).

3.2.2 Mise en œuvre

Comme pour les arbres d'Otter, on se place en la singularité ρ afin d'obtenir des arbres de grande taille.

Le calcul des $T(x^i)$ nécessite alors quelques précautions. En effet, pour des valeurs de x quelconques, on peut, comme pour les partitions, utiliser l'équation (3.1) sous forme tronquée pour obtenir une bonne approximation de $T(x)$. Mais pour des valeurs proches de la singularité, on a une erreur assez importante et il est préférable de procéder différemment. On utilise l'équation

[1] voir le traitement des ensembles étiquetés dans [2]

Algorithme 3.2 $\mathbf{\Gamma}T(x)$: générateur d'arbres généraux

procédure $\mathbf{\Gamma}T_k(x)$
  $\mathrm{T} \leftarrow \Gamma T(x^k)$;
  Retourner $\underbrace{\mathrm{T}\ ;\ \ldots\ ;\ \mathrm{T}}_{\text{k fois}}$
fin procédure

procédure $\mathbf{\Gamma}S_k(x)$
  $p \leftarrow \mathrm{Pois}\left(\frac{T(x^k)}{k}\right)$;
  Retourner $\underbrace{\Gamma T_k(x)\ ;\ \ldots\ ;\ \Gamma T_k(x)}_{p \text{ fois}}$
fin procédure

procédure $\mathbf{\Gamma}S_{\geq 1,k}(x)$
  $p \leftarrow \mathrm{Pois}_{\geq 1}\left(\frac{T(x^k)}{k}\right)$;
  Retourner $\underbrace{\Gamma T_k(x^k)\ ;\ \ldots\ ;\ \Gamma T_k(x^k)}_{p \text{ fois}}$
fin procédure

$k \leftarrow \mathrm{IndiceMaximalPaquet}(x)$
si $k = 0$ **alors**
  Retourner $\square$
sinon
  Retourner $\blacksquare\langle\ \Gamma S_1(x)\ ;\ \ldots\ ;\ \Gamma S_{k-1}(x)\ ;\ \Gamma S_{\geq 1,k}(x)\ \rangle$
fin si

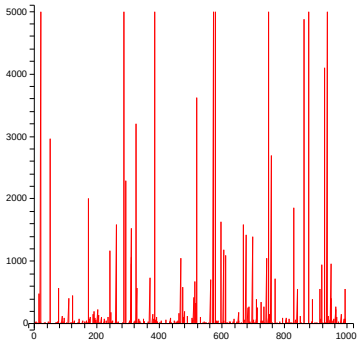


FIG. 3.4 – Histogramme des tailles d'arbres générés par $\Gamma T(\rho)$ sur 1000 tirages.

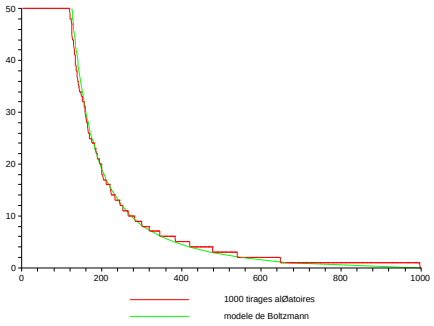


FIG. 3.5 – Distributions en taille (de 1à 50) théorique et expérimentale sur 1000 tirages d'arbres généraux non planaires. Chaque point (x, y) représente un arbre de taille y.

fonctionnelle (3.2) (en négligeant le reste) afin de calculer $T(x)$ récursivement[2] :

$$\begin{aligned} T(x) &= x \exp\left(T(x) + \underbrace{\frac{T(x^2)}{2} + \cdots + \frac{T(x^m)}{m}}_{\substack{S \\ calculé\ récursivement}} + reste \right) \\ &= (xe^S)e^{T(x)} \\ &= \mathrm{CayleyT}(xe^S) \\ &= -\mathrm{LambertW}(-xe^S) \quad \text{[en Maple]}. \end{aligned}$$

La figure 3.4 permet de voir que sur 1000 tirages, on peut obtenir plus d'une dizaine d'arbres de taille supérieure à 5000. La figure 3.5 nous donne la distribution en taille (< 50) obtenue sur 1000 tirages et la figure 3.6 des exemples d'arbres obtenus par $\Gamma T(\rho)$.

3.3 Les circuits série-parallèle

Il s'agit de circuits électriques simples, formés de composants identiques, montés soit en série, soit en parallèle. Ils se décomposent récursivement ainsi : les circuits de type série admettent une unique décomposition en une série de circuits non séries. De même, les circuits parallèles admettent une unique décomposition en un ensemble de circuits non parallèles montés en parallèle. On a donc la décomposition :

[2]se reporter à [3] (chapitre VII.2.), pour plus de détails

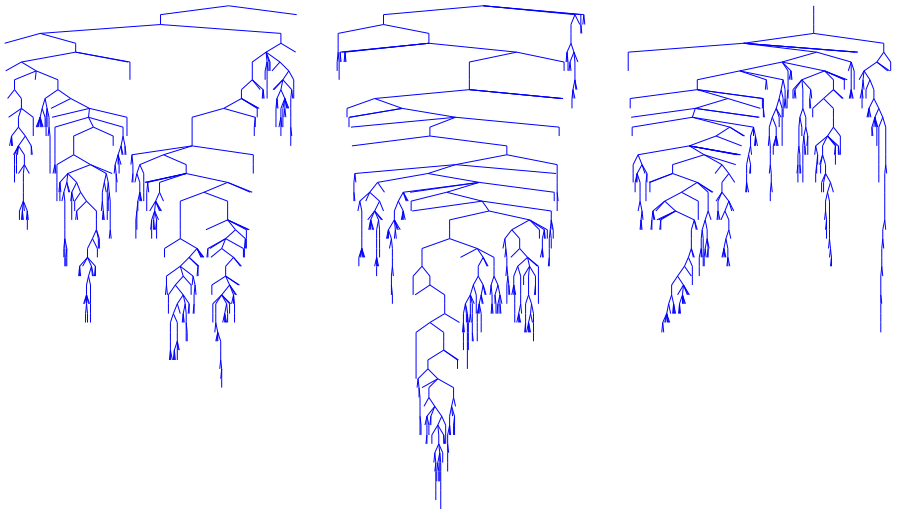


FIG. 3.6 – trois arbres généraux non planaires générés par $\Gamma T(\rho)$

$$\begin{cases} \mathcal{C} &= \mathcal{P} + \mathcal{S} + \mathcal{Z} \\ \mathcal{S} &= \textsc{Seq}_{\geq 2}(\mathcal{P} + \mathcal{Z}) \\ \mathcal{P} &= \textsc{MSet}_{\geq 2}(\mathcal{S} + \mathcal{Z}) \end{cases}$$

On constate que cette décomposition nous donne une structure d'arbre bicolore alterné ressemblant à celle des arbres $\wedge$-$\vee$, à ceci près que les arbres-séries sont planaires (voir figure 3.7), ce qui traduit le fait que les éléments d'une séquence en série sont ordonnés.

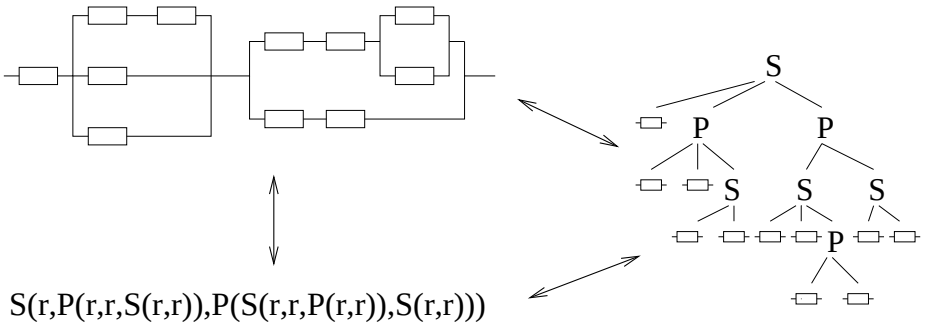


FIG. 3.7 – Un exemple de circuit, un plongement dans le plan de son arbre associé et son expression parenthésée.

On obtient alors les séries génératrices suivantes :

$$C(z) = P(z) + S(z) + z \tag{3.3}$$

$$S(z) = \frac{(P(z)+z)^2}{1-(P(z)+z)} \tag{3.4}$$

$$P(z) = \exp\left(\sum_{k=0}^{\infty} \frac{(S(z)+z)^k}{k}\right) - 1 - (S(z)+z) \tag{3.5}$$

3.3.1 Générateur

La particularité de ce générateur vient des contraintes de taille sur les séquences et les multi-ensembles. Pour engendrer une séquence d'au moins 2 objets, il suffit de décaler la loi géométrique de 2. Pour le multi-ensemble d'au moins 2 objets, on procède par rejet :

- on tire un multi-ensemble quelconque,
- on rejette et on recommence si le nombre de composantes du multi-ensemble tiré n'est pas supérieur à 1.

Comme nous l'avons déjà expliqué au paragraphe 3.2.1, le tirage d'un multi-ensemble consiste déjà à déterminer l'indice maximal de paquet k. Si $k = 0$, on rejette, car alors le nombre d'éléments est nul. Si $k \geq 2$, on sait qu'on peut garder le multi-ensemble que l'on va tirer car il contiendra au moins un paquet de k éléments identiques. Enfin, si $k = 1$, il faut tirer le nombre d'éléments pour l'indice de paquet $k = 1$, ce que l'on fait en tirant une loi de Poisson p non nulle de paramètre $S(x) + x$. Si $p = 1$ le multi-ensemble généré n'aura qu'un élément, donc on le rejette et on recommence. Sinon, $p \geq 2$ et on peut générer le multi-ensemble avec ces valeurs de k et p.

On obtient donc le générateur $\Gamma C(x)$, en posant $\tilde{S}(x) = S(x) + x$ (algorithmes 3.3, 3.4 et 3.5).

Algorithme 3.3 $\Gamma C(x)$: générateur de circuits série-parallèle

si Bern $\left(\frac{x}{C(x)}\right) = 1$ **alors**
  Retourner r
sinon si Bern $\left(\frac{P(x)}{P(x)+S(x)}\right) = 1$ **alors**
  Retourner $\Gamma P(x)$
sinon
  Retourner $\Gamma S(x)$
fin si

3.3.2 Mise en œuvre

De même que pour les autres structures arborescentes, on se place en la singularité ρ de $C(x)$ afin d'avoir de bonnes chances d'obtenir des circuits de

Algorithme 3.4 $\Gamma S(x)$: générateur de circuits série

procédure $\mathbf{\Gamma} S_{aux}(x)$
  si Bern $\left(\frac{x}{P(x)+x}\right) = 1$ **alors**
    Retourner r
  sinon
    Retourner $\Gamma P(x)$
  fin si
fin procédure

$g \leftarrow \text{Geom}_{\geq 2}(x)$
Retourner $\boldsymbol{S}(\underbrace{\Gamma S_{aux}(x)\ ;\ \ldots\ ;\ \Gamma S_{aux}(x)}_{g \text{ fois}})$

grande taille. Il existe plusieurs façons de calculer les $P(x^i)$ et les $S(x^i)$. Voici celle que nous avons choisi.

On pose :

$$\tilde{P}(y) \stackrel{\text{def}}{=} P(y) + y \quad \text{et} \quad \tilde{S}(y) \stackrel{\text{def}}{=} S(y) + y = H(\tilde{P}(y)) + y \tag{3.6}$$

où $H(z) = \frac{z^2}{1-z}$.

L'équation (3.5) implique :

$$\tilde{P}(y) = \frac{1}{1-y} \exp\left(\sum_{i\geq 1} \frac{1}{i} H(\tilde{P}(y^i))\right) - 1 - H(\tilde{P}(y))$$

D'où $\tilde{P}(y)$ est un nombre X qui vérifie :

$$X = \frac{1}{1-y} \exp\left(\sum_{i\geq 2} \frac{1}{i} H(\tilde{P}(y^i))\right) \exp\left(H(X)\right)) - 1 - H(X)) \tag{3.7}$$

On introduit la série génératrice $\hat{P}(y)$ définie par l'équation suivante :

$$\hat{P}(y) \stackrel{\text{def}}{=} \frac{1}{1-y} \exp\Big(H(\hat{P}(y))\Big) - 1 - H(\hat{P}(y)) \tag{3.8}$$

Et pour chaque y, on introduit $\tilde{y}$ tel que :

$$\frac{1}{1-\tilde{y}} \stackrel{\text{def}}{=} \frac{1}{1-y} \exp\left(\sum_{i\geq 2} \frac{1}{i} H(\tilde{P}(y^i))\right) \tag{3.9}$$

Donc :

Algorithme 3.5 $\Gamma P(x)$: générateur de circuits parallèle

procédure $\mathbf{\Gamma}\tilde{S}_k(x)$
  si Bern $\left(\frac{x^k}{S(x^k)+x^k}\right) = 1$ **alors**
    Retourner r
  sinon
    S $\leftarrow \Gamma S(x^k)$;
    Retourner $\underbrace{\text{S} \ ; \ \ldots \ ; \ \text{S}}_{k \text{ fois}}$
  fin si
fin procédure

procédure $\mathbf{\Gamma}V_k(x)$
  $p \leftarrow$ Pois $\left(\frac{\tilde{S}(x^k)}{k}\right)$;
  Retourner $\underbrace{\Gamma\tilde{S}_k(x) \ ; \ \ldots \ ; \ \Gamma\tilde{S}_k(x)}_{p \text{ fois}}$
fin procédure

procédure $\mathbf{\Gamma}V_{\geq 1,k}(x)$
  $p \leftarrow \text{Pois}_{\geq 1}\left(\frac{\tilde{S}(x^k)}{k}\right)$;
  Retourner $\underbrace{\Gamma\tilde{S}_k(x) \ ; \ \ldots \ ; \ \Gamma\tilde{S}_k(x)}_{p \text{ fois}}$
fin procédure

$k \leftarrow 1$;
$p \leftarrow 1$;
tant que $k = 0$ **ou** $(k = 1$ **et** $p = 1)$ **faire**
  $k \leftarrow \text{IndiceMaximalPaquet}_{\geq 1}(x)$;
  $p \leftarrow \text{Pois}_{\geq 1}(\tilde{S}(x))$;
fin tant que

si $k = 1$ **alors**
  Retourner $\boldsymbol{P}(\underbrace{\Gamma\tilde{S}_1(x) \ ; \ \ldots \ ; \ \Gamma\tilde{S}_1(x)}_{p \text{ fois}})$
sinon
  Retourner $\boldsymbol{P}(\ \Gamma V_1(x) \ ; \ \ldots \ ; \ \Gamma V_{k-1}(x) \ ; \ \Gamma V_{\geq 1,k}(x)\ )$
fin si

$$\begin{aligned}\hat{P}(\tilde{y}) &= \frac{1}{1-\tilde{y}}\exp\Big(H(\hat{P}(\tilde{y}))\Big) - 1 - H(\hat{P}(\tilde{y})) \qquad (3.10)\\ &= \frac{1}{1-y}\exp\left(\sum_{i\geq 2}\frac{1}{i}H(\tilde{P}(y^i))\right)\exp\Big(H(\hat{P}(\tilde{y}))\Big) - 1 - H(\hat{P}(\tilde{y}))\end{aligned}$$

Donc $\hat{P}(\tilde{y})$ vérifie également l'équation (3.7). On peut montrer formellement que $\hat{P}(\tilde{y})$ et $\tilde{P}(y)$ son égales en tant que séries génératrices en la variable y car elle vérifient l'équation (3.7) et elles ont les mêmes coefficients initiaux. On en déduit que, pour tout y, les nombres $\hat{P}(\tilde{y})$ et $\tilde{P}(y)$ sont égaux.
On déduit de ce qui précède une méthode récursive rapide pour l'évaluation de $\tilde{P}(y)$:

1. calculer la valeur $\tilde{y}$ associée à y selon l'équation (3.9) : le calcul de $\tilde{y}$ se fait par des appels récursifs à des évaluations de $\tilde{P}(y^i)$ pour $i \geq 2$.
2. évaluer $\hat{P}(\tilde{y})$ en utilisant l'équation (3.8) . On obtient une très bonne précision d'estimation en tronquant la récursivité à une profondeur $n = 20$, i.e. en faisant l'approximation $\tilde{P}(y^i) = 0$ pour $i \geq 20$

Enfin, on pose :

$$G(\hat{P}, \tilde{y}) = \frac{1}{1-\tilde{y}}\exp\Big(H(\hat{P})\Big) - 1 - H(\hat{P}) - \hat{P}$$

Et on obtient $\tilde{\rho}$, la singularité de $\hat{P}$ ainsi que la valeur de $\hat{P}(\tilde{\rho})$ en résolvant le système :

$$\left\{\frac{\partial G}{d\hat{P}}(\hat{P}, \tilde{\rho}) = 0 \ ; \ G(\hat{P}, \tilde{\rho}) = 0\right)$$

$\hat{P}(\tilde{\rho})$ nous donne donc la valeur de $\tilde{P}$ en sa singularité ρ, et comme on a une procédure rapide d'évaluation de $\tilde{P}$, on peut aisément évaluer ρ (par dichotomie, par exemple).

En se plaçant en la singularité, on obtient, avec une bonne probabilité, des circuits de taille importante. Les figures 3.8 et 3.9 présentent des exemples de circuits de taille respectivement 100 et ~ 500 obtenus par $\Gamma C(\rho)$. Comme pour les arbres, on peut alors adapter ce générateur pour qu'il ne dépasse pas un certain niveau d'emboîtement des circuits.

3.4 Conclusion

Nous disposons désormais d'un algorithme efficace (voir Algorithme générique 1) de génération de multi-ensembles sous modèle de Boltzmann, ainsi que certaines de ses variantes avec contraintes de cardinalité. Les générateurs qu'il fournit ont une complexité $O(n)$ dans le pire des cas. Ces résultats se traduisent par les

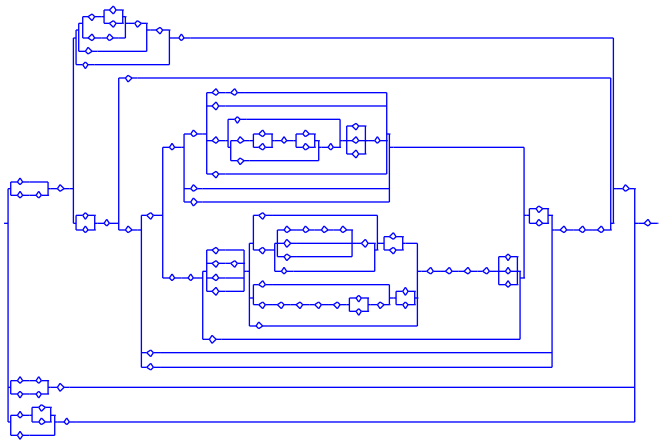


FIG. 3.8 – Un circuit de taille 100 généré par $\Gamma C(\rho)$

théorèmes 3.1 et 3.2. Dans la suite de cette étude, nous allons voir comment utiliser cette construction afin de mettre sur pied des générateurs d'ensembles sans répétitions.

Théorème 3.1 *Si on possède un générateur de Boltzmann pour $\mathcal{A}$, alors, l'algorithme générique 1 fournit un générateur aléatoire de Boltzmann pour $\mathcal{M} = \textsc{MSet}(\mathcal{A})$. Cet algorithme produit un objet $\mu \in \mathcal{M}$ en temps linéaire en la taille de μ.*

Théorème 3.2 *Pour les classes combinatoires suivantes, pour des valeurs du paramètre x optimisées, et en utilisant une méthode par rejet, on peut obtenir des générateurs de Boltzmann en taille approchée avec les complexités moyennes suivantes :*

Partitions d'entier (nombre de sommant borné) de taille n	$O(\sqrt{n})$
Partitions d'entier de taille n	$O(\sqrt{n})$
Arbres d'Otter de taille n	$O(n)$
Arbres non planaires de taille n	$O(n)$
Circuits série-parallèle de taille n	$O(n)$

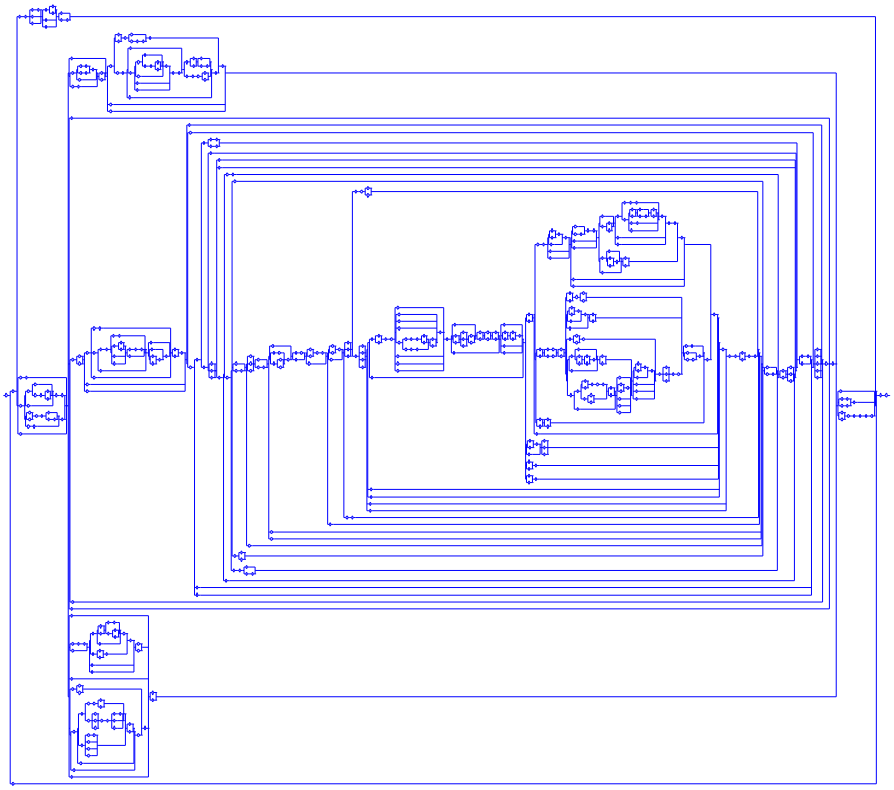


FIG. 3.9 – Un circuit de taille ~ 500 généré par $\Gamma C(\rho)$

– Série génératrice

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

– Générateur de Boltzmann pour $\mathcal{M}$

1. tirer k, la plus grande taille de paquet, on utilise la procédure suivante :
   IndiceMaximalPaquet(x)
   $U \leftarrow \text{random}(0,1)$;
   $V \leftarrow \log \frac{1}{U}$;
   $k \leftarrow 0$;
   $p \leftarrow \log M(x)$;
   tant que $V < p$ **faire**
   $k \leftarrow k+1$;
   $p = p - A(x^k)/k$;
   fin tant que
   Retourner k ;
2. tirer, pour j variant de 1 à k, des ensembles de paquets de j copies d'un élément de $\mathcal{A}$, on utilise la procédure suivante :
   j_Copies(x)
   $A \leftarrow \Gamma A(x^j)$;
   Retourner ($\underbrace{A \ ; \ \ldots \ ; \ A}_{j \ fois}$) ;
3. renvoyer la séquence des k ensembles obtenus précédement.

$\Gamma M(x)$: générateur de multi-ensembles

$k \longleftarrow$ IndiceMaximalPaquet(x) ;
pour j **de** 1 **à** $k-1$ **faire**
$p_j \longleftarrow \text{Pois}\left(\frac{A(x^j)}{j}\right)$
$ens_j \longleftarrow (\underbrace{j\_Copies(x), \ldots, j\_Copies(x)}_{p_j \ fois})$
fin pour
$p_k \longleftarrow \text{Pois}_{\geq 1}\left(\frac{A(x^k)}{k}\right)$
$ens_k \longleftarrow (\underbrace{k\_Copies(x), \ldots, k\_Copies(x)}_{p_k \ fois})$
Retourner (ens_1 ; … ; ens_k)

Algorithme générique 1: $\mathcal{M} = \mathbf{MSet}(\mathcal{A})$

Chapitre 4

Générateurs de Boltzmann pour des ensembles sans répétitions

On va implanter des générateurs de Boltzmann pour tirer des ensembles sans répétitions d'objets d'une classe combinatoire.

Tout multi-ensemble d'objets de $\mathcal{A}$ se décompose de façon unique en un ensemble d'objets de $\mathcal{A}$ et un multi-ensemble de paquets de deux objets identiques de $\mathcal{A}$:

$$\textsc{MSet}(\mathcal{A}) = \textsc{PSet}(\mathcal{A}) \times \textsc{MSet}(\mathcal{A}^{(2)}) \tag{4.1}$$

On considère un multi-ensemble m d'objets et on lui associe un couple formé d'un ensemble sans répétitions $p \in \textsc{PSet}(\mathcal{A})$ et d'un multi-ensemble d'objets dédoublés de $\mathcal{A}$, $m_2 \in \textsc{MSet}(\mathcal{A}^{(2)})$. On détermine l'appartenance à p ou m_2 d'un objet $\gamma \in \mathcal{A}$ selon sa multiplicité dans m : l'objet γ appartient à p si, et seulement si, il apparaît un nombre impair de fois dans m.
On peut donc aisément obtenir un ensemble sans répétitions à partir d'un multi-ensemble en ne gardant que les objets dont le nombre d'occurrences est impair dans le multi-ensemble.
Cela implique que si l'on dispose d'un générateur de Boltzmann pour m, alors par extraction, on dipose d'un générateur de Boltzmann pour p [*Preuve :* Si $\mathcal{C} \cong \mathcal{A} \times \mathcal{B}$, alors $\Gamma C(x) = \Gamma A(x) \times \Gamma B(x)$].
Nous allons présenter ici 3 générateurs :

- Un générateur de langages finis de mots binaires.
- Un générateur d'arbres généraux non planaires enracinés sans jumeaux. Nous pourrons alors comparer leur forme avec celle des arbres sans restriction.
- Un générateur de partitions d'entier en sommants distincts. Nous présenterons en fait 2 générateurs différents afin de vérifier l'efficacité et la pertinence de la construction que nous proposons pour les ensembles.

4.1 Les langages finis

Les langages finis sur un alphabet $\mathcal{A}$ constituent une classe d'objets combinatoires dont la fonction taille est le nombre total de lettres de tous les mots du langage. Un mot est une séquence non nulle de lettres de $\mathcal{A}$ et un langage est un ensemble sans répétitions de mots sur $\mathcal{A}$. On a donc la décomposition suivante :

$$\mathcal{L} = \textsc{PSet}(\textsc{Seq}_{\geq 1}(\mathcal{A}))$$

On se place sur l'alphabet binaire $\{a, b\}$. On a alors les séries génératrices équivalentes :

$$\begin{aligned} L(z) &= \prod_{\gamma \in \mathcal{L}} (1 + z^{|\gamma|}) \\ &= \prod_{n \geq 1} (1 + z^n)^{2^n} \\ &= \exp\left(\sum_{k=1}^{\infty} \frac{(-1)^{k-1}}{k} \frac{2z^k}{1 - 2z^k} \right) \end{aligned} \tag{4.2}$$

4.1.1 Générateur

D'après le tableau 1.1, on construit facilement un générateur de mots non vides sur l'alphabet $\{a, b\}$ sachant que la série génératrice est $W(z) = \frac{2z}{1-2z}$ (algorithme 4.1).

Algorithme 4.1 $\Gamma W(x)$: générateur de mots binaires

procédure $\Gamma Lt(x)$
  si Bern$(1/2) = 1$ **alors**
    Retourner a ;
  sinon
    Retourner b ;
  fin si
fin procédure

$g \leftarrow \text{Geom}_{\geq 1}(2x)$;
Retourner ($\underbrace{\Gamma Lt(x);\ \ldots\ ;\ \Gamma Lt(x)}_{g \text{ fois}}$)

Un générateur de Boltzmann pour un ensemble de mots binaires s'obtient alors en fabriquant, dans un premier temps, un générateur de multi-ensembles de mots et, suivant l'équation (4.1), en ne gardant qu'une seule occurrence des mots qui sont répétés un nombre impair de fois dans le multi-ensemble.

On a donc la procédure MSet2PSet de transformation d'un multi-ensemble en un ensemble (procédure 4.1).

Procédure 4.1 MSet2PSet(M)

$P := \emptyset$;
pour $\gamma \in M$ **faire**
  $c_\gamma :=$ nombre d'occurrences de γ dans M ;
  si c_γ modulo $2 = 1$ **alors**
    $P := P \cup \{\gamma\}$;
  fin si
fin pour
Retourner P ;

On remarque qu'il est possible d'optimiser très simplement $\Gamma L(x)$ en ne générant ni les paquets d'indice pair qui seraient inévitablement jetés par la suite, ni les répétitions d'objets dues aux paquets d'indice impair.

Le générateur de langages finis a donc la structure donnée par l'algorithme 4.2 (nous ne détaillons pas la fonction IndiceMaximalPaquet(x), le principe est le même que pour les générateurs présentés aux chapitres précédents (voir procédures 2.2 et 3.1)) :

Algorithme 4.2 $\Gamma L(x)$: générateur de langages finis sur un alphabet binaire

procédure $\Gamma L_k(x)$
  si k est pair **alors**
    Retourner $\emptyset$
  sinon
    $p \leftarrow \text{Pois}\left(\frac{W(x^k)}{k}\right)$;
    Retourner ($\underbrace{\Gamma W(x^k);\ \ldots\ ;\ \Gamma W(x^k)}_{p \text{ fois}}$)
  fin si
fin procédure

procédure $\Gamma L_{\geq 1,k}(x)$
  si k est pair **alors**
    Retourner $\emptyset$
  sinon
    $p \leftarrow \text{Pois}_{\geq 1}\left(\frac{W(x^k)}{k}\right)$;
    Retourner ($\underbrace{\Gamma W(x^k);\ \ldots\ ;\ \Gamma W(x^k)}_{p \text{ fois}}$)
  fin si
fin procédure

$k \leftarrow \text{IndiceMaximalPaquet}(x)$;
$M \leftarrow (\Gamma L_1(x)\ ;\ \ldots\ ;\ \Gamma L_{k-1}(x)\ ;\ \Gamma L_{\geq 1,k}(x)\ )$;
Retourner $\text{MSet2PSet}(M)$;

4.1.2 Mise en œuvre

Implémentation de $MSet2PSet(M)$

Nous avons cherché à optimiser l'élimination des répétitions. Pour déterminer si un objet apparaît un nombre pair ou impair de fois, il faut pouvoir compter les occurences de cet objet dans le multi-ensemble, et donc pouvoir comparer les objets qu'il contient. Pour cela, nous avons utilisé une particularité de MAPLE : le partage des expressions en mémoire. En effet, chaque expression MAPLE possède une 'étiquette' et 2 expressions ou sous-expressions identiques ne sont stockées qu'une seule fois, avec la même étiquette, grâce à une méthode de hachage. Comparer des expressions MAPLE revient donc à comparer des entiers. Une comparaison se fait alors en temps constant, et non pas linéaire en la taille de l'expression. De plus, il est possible d'indicer un tableau MAPLE avec n'importe quelle expression, ce qui nous permet de fabriquer, en une seule passe sur le multi-ensemble, une table de compteurs d'occurrences des objets qui le composent. La suppression des répétitions est donc linéaire en la taille du multi-ensemble, et $MSet2PSet(s)$ s'écrit, en MAPLE :

```
MSet2PSet :=proc(ms)
        for object in ms do cpt[object] :=0 ; od ;
        for object in ms do cpt[object] :=cpt[object]+1 ; od ;
        s :=[ seq( `if`(cpt[o] mod 2 = 1, o, NULL),
                    o = map( op, [indices(cpt)] )     ) ] ;
end ;
```

Choix du paramètre x et résultats

La série $L(z)$ (équation (4.2)) possède une singularité évidente en $\rho = \frac{1}{2}$. La figure 4.1 nous montre que nous avons des distributions de Boltzmann de type "bumpy"[1] pour des valeurs de x plus ou moins proches de ρ. En choisissant une valeur de x assez proche de 0.5, on peut donc facilement obtenir des objets de grande taille. Sur 10000 tirages aléatoires de $\Gamma L(0.45)$, on obtient, par exemple, la distribution présentée à la figure 4.2, conforme à la distribution théorique attendue sous le modèle de Boltzmann.

4.2 Les arbres généraux non planaires enracinés sans jumeaux

Ce sont les mêmes arbres que ceux présentés à la section 3.2, à ceci près que chaque nœud a un ensemble (et non plus un multi-ensemble) de fils. Il ne peut donc pas y avoir de jumeaux dans un arbre. La décomposition devient :

$$\tilde{\mathcal{T}} = \mathcal{Z} \times \mathrm{PSET}(\tilde{\mathcal{T}})$$

[1] voir [2]

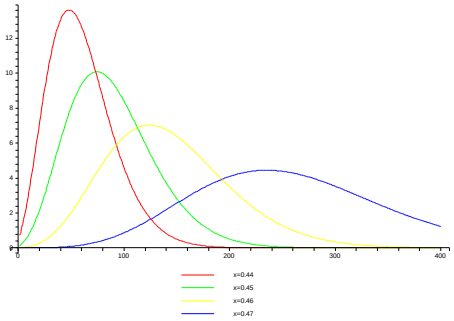


FIG. 4.1 – Distribution des tailles sous le modèle de Boltzmann pour les langages finis, pour des valeurs du paramètre x s'approchant de la singularité ρ de $L(z)$.

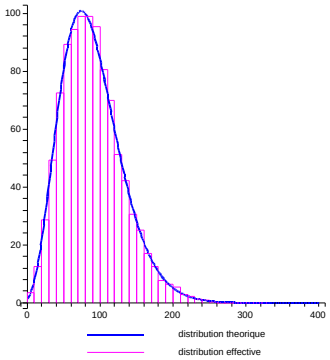


FIG. 4.2 – Distribution en taille (théorique et expérimentale) sous le modèle de Boltzmann, pour $x = 0.45$

Cette décomposition nous donne la série génératrice suivante :

$$\begin{aligned}
\tilde{T}(z) &= z \prod_{\gamma \in \tilde{T}} (1 + z^{|\gamma|}) \\
&= z \prod_{n \geq 1} (1 + z^n)^{\tilde{T}_n} \qquad (4.3) \\
&= z \exp\left(\sum_{k=1}^{\infty} \frac{(-1)^{k-1}}{k} \tilde{T}(z^k) \right) \qquad (4.4)
\end{aligned}$$

4.2.1 Générateur

Comme précédemment, pour générer les fils de la racine, on commence par générer un multi-ensemble d'arbres sans jumeaux récursivement, puis on supprime les répétitions. Le code de base de $\Gamma\tilde{T}(x)$ est donc similaire à celui du générateur d'arbres donné au chapitre précédent. On l'optimise pour la génération d'ensembles en évitant de générer les répétitions dues au paquets, comme pour $\Gamma L(x)$. Enfin, si le multi-ensemble obtenu est non vide, on supprime les répétitions en utilisant MSet2PSet(M) (algorithme 4.3).

4.2.2 Mise en œuvre et résultats

Comme pour les arbres classiques, on se place en la singularité ρ de $\tilde{T}(z)$ afin de pouvoir générer des arbres de grande taille. Pour calculer ρ et les $\tilde{T}(x^i)$ on utilise la méthode de [3] déjà mentionnée plus haut. Sur 1000 tirages de $\Gamma\tilde{T}(\rho)$, on obtient la distribution en taille présentée à la figure 4.3.

Algorithme 4.3 $\Gamma\tilde{T}(x)$: générateur d'arbres sans jumeaux

procédure $\Gamma\tilde{T}_k(x)$
  si k est pair **alors**
    Retourner $\emptyset$
  sinon
    $p \leftarrow \text{Pois}\left(\frac{T(x^k)}{k}\right)$;
    Retourner $\underbrace{\Gamma T(x^k)\ ;\ \ldots\ ;\ \Gamma T(x^k)}_{p \text{ fois}}$
  fin si
fin procédure

procédure $\Gamma\tilde{T}_{\geq 1,k}(x)$
  si k est pair **alors**
    Retourner $\emptyset$
  sinon
    $p \leftarrow \text{Pois}_{\geq 1}\left(\frac{T(x^k)}{k}\right)$;
    Retourner $\underbrace{\Gamma T(x^k)\ ;\ \ldots\ ;\ \Gamma T(x^k)}_{p \text{ fois}}$
  fin si
fin procédure

$k \leftarrow \text{IndiceMaximalPaquet}(x)$;
si $k = 0$ **alors**
  Retourner $\square$
sinon
  $M \leftarrow (\ \Gamma\tilde{T}_1(x)\ ;\ \ldots\ ;\ \Gamma\tilde{T}_{k-1}(x)\ ;\ \Gamma\tilde{T}_{\geq 1,k}(x)\ )$;
  Retourner $\blacksquare\langle\ \text{MSet2PSet}(M)\ \rangle$;
fin si

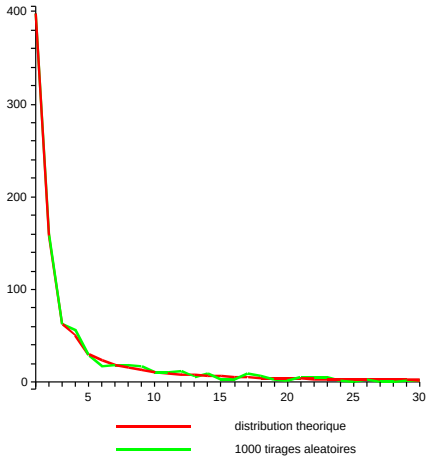


FIG. 4.3 – Arbres généraux non planaires enracinés sans jumeaux : distribution (effective et théorique) des tailles pour 1000 tirages de $\Gamma\tilde{T}(\rho)$.

La figure 4.5 nous donne plusieurs exemples d'arbres de différentes tailles obtenus par $\Gamma\tilde{T}(\rho)$. On observe que la forme générale des arbres ainsi obtenue est "assez proche" de celle des arbres sans restrictions. Cela vient de ce qu'en pratique, on a rarement de répétitions de sous-arbres de taille importante dans un arbre classique. De ce fait, c'est au niveau de la frange que l'on observe certaines différences car on ne peut pas avoir une fratrie de feuilles dans les arbres sans jumeaux. Ainsi, on peut voir des "coulures" sur la frange qui font que l'on a, en moyenne, moins de feuilles sur un arbre sans jumeaux que sur un arbre classique (voir figure 4.4). Une étude des séries bivariées $T(z,u)$ et $\tilde{T}(z,u)$ où u marque le nombre de feuilles permettrait de confirmer ces observations.

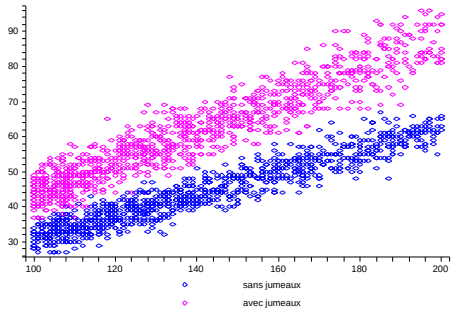


FIG. 4.4 – Nombre de feuilles en fonction de la taille pour des arbres (avec ou sans jumeaux) dont la taille est comprise entre 100 et 200.

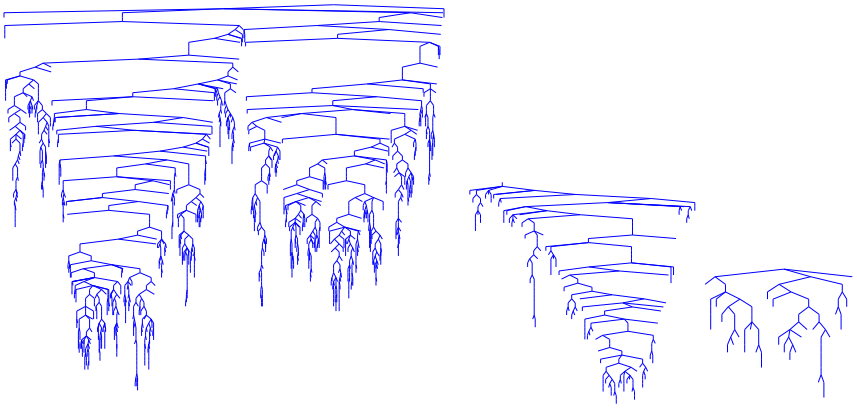


FIG. 4.5 – Arbres généraux non planaires enracinés sans jumeaux générés par $\Gamma\tilde{T}(\rho)$. Tailles de gauche à droite : 8643, 288, 88.

4.3 Partitions d'entier en sommants distincts

Au lieu de considérer des multi-ensembles d'entiers, on étudie désormais des ensembles sans répétitions, i.e., il ne peut pas y avoir deux sommants de même taille dans une partition. La décomposition qui correspond est :

$$\tilde{\mathcal{P}} = \textsc{PSet}(\mathcal{I}),\ \ \mathcal{I} = z \times \textsc{Seq}(z)$$

Ce qui donne deux expressions équivalentes pour la série génératrice de $\tilde{\mathcal{P}}$:

$$\tilde{P}(z) = exp\left(\sum_{k=1}^{\infty} \frac{(-1)^{k-1}}{k} \frac{z^k}{(1-z^k)}\right) \tag{4.5}$$

$$= \prod_{n=1}^{\infty}(1+z^n) \tag{4.6}$$

4.3.1 Construction par produit.

Générateur

Dans un premier temps, on cherche à construire un générateur de façon directe, basé sur l'équation (4.6), afin de comparer les résultats obtenus avec un générateur qui fabrique un multi-ensemble et supprime les copies.

L'équation (4.6) nous suggère le générateur qui consiste à déterminer, pour chaque taille de sommant, si on inclut ou non le sommant dans la partition que l'on est en train de construire.

Il faut pour cela commencer par tirer k, la taille du plus grand sommant (de manière à simuler un tirage infini). On utilise le même schéma que pour la procédure 2.1 (voir section 2.3.1). Ensuite, pour chaque taille i jusqu'à $k-1$,

on détermine si l'on prend ou non le sommant de taille i en tirant une loi Bern $\left(\frac{1}{1+x^i}\right)$. On impose enfin un sommant de taille k, en accord avec le tirage de k. Cela nous donne le générateur $\Gamma\tilde{P}_1(x)$ (algorithme 4.4).

Algorithme 4.4 $\Gamma\tilde{P}_1(x)$: générateur de partitions d'entiers en sommants distincts (produit)

procédure $\Gamma S_k(x)$
  si Bern$(\frac{1}{1+x^k}) = 1$ **alors**
    Retourner $\emptyset$;
  sinon
    Retourner k ;
  fin si
fin procédure

$k \leftarrow$ PlusGrandSommant(x) ;
Retourner ($\Gamma S_1(x)$; ... ; $\Gamma S_{k-1}(x)$; k)

Mise en œuvre

On choisit le paramètre x_0 du générateur de façon à avoir "de bonnes chances" d'obtenir des partitions de taille N. Par un calcul similaire à celui fait pour les partitons générales, on trouve que $x_N = 1 - \pi/\sqrt{12N}$ est une bonne approximation de x_0.

On peut observer, sur le diagramme de la figure 4.7, la distribution en taille obtenue avec $\Gamma\tilde{P}_1(0.95)$

On a pu observer, en tirant une partition avec $\Gamma\tilde{P}_1(1 - \pi/\sqrt{12.10^5})$, le profil estimé dans [9] d'une partition aléatoire d'entier en sommants distincts (figure 4.6).

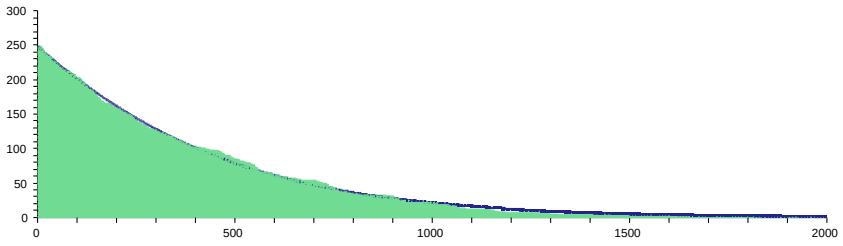


FIG. 4.6 – Partition aléatoire de taille $\sim 10^5$ générée par $\Gamma\tilde{P}_1(1 - \pi/\sqrt{12.10^5})$ et son profil théorique.

4.3.2 Construction par multi-ensemble

Générateur

Comme pour les arbres, on peut reprendre le générateur de partitions sans restrictions et supprimer (de façon optimisée) les répétitions de sommants. On a alors le générateur $\Gamma\tilde{P}_2(x)$ (algorithme 4.5).

Algorithme 4.5 $\Gamma\tilde{P}_2(x)$: générateur de partitions d'entiers en sommants distincts (exponentielle)

---

procédure $\Gamma S_k(x)$
  si k est pair **alors**
    Retourner $\emptyset$
  sinon
    $p \leftarrow \text{Pois}\left(\frac{x^k}{k(1-x^k)}\right)$;
    Retourner ($\underbrace{\Gamma I_k(x^k)\ ;\ \ldots\ ;\ \Gamma I_k(x^k)}_{p \text{ fois}}$)
  fin si
fin procédure

procédure $\Gamma S_{\geq 1,k}(x)$
  si k est pair **alors**
    Retourner $\emptyset$
  sinon
    $p \leftarrow \text{Pois}_{\geq 1}\left(\frac{x^k}{k(1-x^k)}\right)$;
    Retourner ($\underbrace{\Gamma I_k(x^k)\ ;\ \ldots\ ;\ \Gamma I_k(x^k)}_{p \text{ fois}}$)
  fin si
fin procédure

$k \leftarrow \text{IndiceMaximalPaquet}(x)$;
$M \leftarrow (\ \Gamma S_1(x)\ ;\ \ldots\ ;\ \Gamma S_{k-1}(x)\ ;\ \Gamma S_{\geq 1,k}(x)\ )$;
Retourner $\text{MSet2PSet}(M)$;

Mise en œuvre et comparaison de $\Gamma\tilde{P}_1(x)$ et $\Gamma\tilde{P}_2(x)$.

Comme avec $\Gamma\tilde{P}_1(x)$, on obtient une distribution en taille conforme au modèle de Boltzmann (voir figure 4.8).

Nous avons comparé expérimentalement l'efficacité de $\Gamma\tilde{P}_1(x)$ et $\Gamma\tilde{P}_2(x)$, en effectuant des séries de 10000 tirages pour différentes valeurs du paramètre x (dans les deux cas, nous avons pris soin de pré-calculer les valeurs de séries génératrices utilisées par les générateurs de Boltzmann). Les résultats obtenus sont présentés à la figure 4.9. On constate que, pour générer des partitions de grande taille, $\Gamma\tilde{P}_2(x)$ est un peu plus efficace que $\Gamma\tilde{P}_1(x)$ bien que la méthode

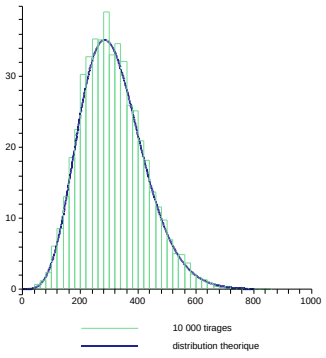
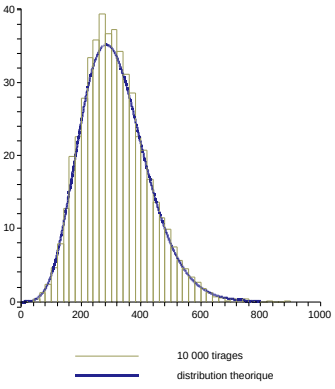


FIG. 4.7 – Distribution en taille de partition générées par $\Gamma\tilde{P}_1(0.95)$



.pdf

FIG. 4.8 – Distribution en taille de partition générées par $\Gamma\tilde{P}_2(0.95)$

de génération choisie pour $\Gamma\tilde{P}_1(x)$ soit directe et ne nécessite pas l'utilisation de lois de Poisson.

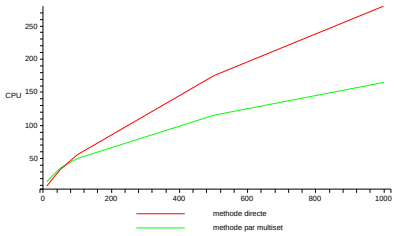


FIG. 4.9 – Comparaison du temps CPU utilisé pour des séries de 10 000 tirages, pour différentes tailles de partitions, de $\Gamma\tilde{P}_1(x)$ et $\Gamma\tilde{P}_2(x)$

Cette relative inefficacité vient d'une part du fait que, par la méthode directe, on tire beaucoup plus de lois de Bernoulli qu'on ne génère effectivement de sommants, i.e., beaucoup de tirages de Bernoulli renvoient 0 (voir le profil moyen d'une partition en sommants distincts : figure 4.10).

D'autre part, nous avons estimé le rapport entre la taille d'une partition classique et la taille de la partition en sommant distincts induite par celle-ci, afin d'apprécier l'efficacité de $\Gamma\tilde{P}_2(x)$. Nous avons donc pu observer que le rapport moyen entre la taille d'une partition en sommants distincts et la taille de la partition générale à partir de laquelle nous l'avons générée était de l'ordre d'une constante C valant environ $\frac{1}{2}$ (voir figure 4.11). La génération du multi-ensemble a donc un coût $O(\sqrt{n})$ (cf. section 2.3.2), où n est la taille de la partition en sommants distincts générée. Enfin, pour calculer les multiplicités,

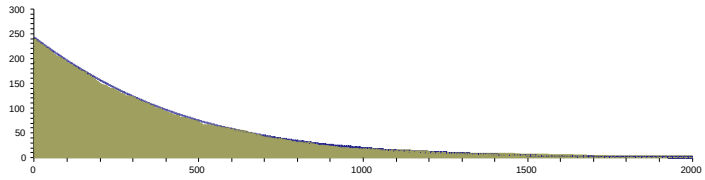


FIG. 4.10 – Partition aléatoire de taille $\sim 10^5$ générée par $\Gamma\tilde{P}_1(1-\pi/\sqrt{12.10^5})$ et son profil théorique.

il faut parcourir toute la liste des tailles de sommant, dont le nombre est de l'ordre de $\sqrt{n}$. On a donc un générateur de complexité $O(\sqrt{n})$

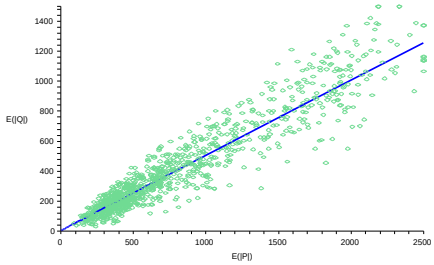


FIG. 4.11 – Rapport (théorique et effectif) entre la taille d'une partition classique et la taille de sa partition en sommants distincts induite.

4.4 Discussion sur la complexité du générateur

Nous venons de voir que dans le cas des partitions, le rapport entre la taille du multi-ensemble et la taille de son ensemble induit est constant. On cherche à savoir si ce cas de figure se généralise à toutes les classes combinatoires.

Soit $\mathcal{P} = \text{PSET}(\mathcal{C})$, la classe des ensembles d'objets de $\mathcal{C}$ et $P(z)$ sa série génératrice associée. Soit $\mathcal{F} = \text{MSET}(\mathcal{C})$ la classe des multi-ensembles d'objets de $\mathcal{C}$ et $F(z)$ sa série génératrice associée. L'espérance de la taille des objets de $\mathcal{P}$ (resp. $\mathcal{F}$) engendrés sous modèle de Boltzmann de paramètre x est $\sum_k x^k C'(x^k)$ (resp. $\sum_k (-1^k) x^k C'(x^k)$). Pour savoir si notre algorithme est efficace, on cherche à connaître la nature du rapport entre ces deux espérances. Dans le cas où $C(z)$ possède une singularité $\rho < 1$, on utilise le lemme 4.1 pour évaluer le rapport des espérances.

- soit $C'(x) = \infty$ et le rapport est de 1,
- soit $C'(x) < \infty$ et le rapport est une constante.

Notre algorithme a donc une complexité $O(n)$ en moyenne dans les deux cas.

Dans le cas où $\rho = 1$, une analyse plus fine serait nécessaire pour déterminer le comportement du rapport des tailles.

Lemme 4.1 *On pose*

$$f(z) = \sum_{n\geq 1} f_n z^n \quad et \quad f'(z) = \sum_{n\geq 1} n f_n z^{n-1}$$

avec f à coefficients positifs, de rayon de convergence $\rho < 1$.
On a alors :

$$S = \sum_{\boldsymbol{k}\geq 2} \rho^k f'(\rho^k) < \infty$$

Preuve :
Soit R tel que $\rho^2 < R < \rho$, d'après Cauchy, on a $f_n \geq C.R^{-n}$. On a alors :

$$f(x) \leq C.\sum_{n=1}^{\infty} x^n R^{-n} \leq C.\frac{x/r}{1-x/R} = C.\frac{x}{R-x}, \quad 0 \leq x \leq R$$

$$\sum_{k\leq 2}^{\infty} f(x^k) \leq C.\sum_{k=2}^{\infty} \frac{x^k}{Rx^k} \leq C.\frac{1}{R-\rho^2}\sum_{k=2}^{\infty} x^k \leq \frac{C.\rho^2}{(R-\rho^2)(1-\rho^2)}, \quad x \leq \rho$$

On remarque également, qu'il est possible de calculer, pour un générateur donné, l'espérance du surcoût dû à la génération du multi-ensemble, en calculant l'espérance de la taille, sous le modèle de Boltzmann, du multi-ensemble de paires qui est rejeté.

4.5 Conclusion

Nous disposons désormais d'un algorithme simple et efficace (voir Algorithme générique 2) de génération d'ensembles sans répétitions sous modèle de Boltzmann. Les générateurs qu'il fournit ont une complexité $O(n)$ en moyenne, dans le cas où le rayon de convergence de la série est inférieur à 1. Ces résultats se traduisent par les théorèmes 4.1 et 4.2. .

Théorème 4.1 *Si on possède un générateur de Boltzmann pour $\mathcal{A}$, alors, l'algorithme générique 2 fournit un générateur aléatoire de Boltzmann pour $\mathcal{S} =$ PSET($\mathcal{A}$). Cet algorithme produit un objet $\mu \in \mathcal{M}$ en temps linéaire en la taille de μ, dans les cas où le rayon de convergence de $A(z)$ est < 1.*

Théorème 4.2 *Pour les classes combinatoires suivantes, pour des valeurs du paramètre x optimisées, et en utilisant une méthode par rejet, on peut obtenir des générateurs de Boltzmann en taille approchée avec les complexités moyennes suivantes :*

Langages finis de taille n	$O(n)$
Arbres non planaires sans jumeaux de taille n	$O(n)$
Partitions d'entier en sommants distincts, de taille n	$O(\sqrt{n})$

– Série génératrice

$$C(z) = \exp\left(\sum_{k\geq 1} \frac{(-1)^{k-1}}{k} A(z^k)\right)$$

– Générateur de Boltzmann pour $\mathcal{S}$

1. engendrer un multi-ensemble d'objets de $\mathcal{A}$
2. en extraire l'ensemble d'objets de $\mathcal{A}$ induit (procédure MSet2PSet) à partir de :

$$\textsc{MSet}(\mathcal{A}) = \textsc{PSet}(\mathcal{A}) \times \textsc{MSet}(\mathcal{A}^{(2)})$$

MSet2PSet(M)

$P \leftarrow \emptyset$;
pour $\gamma \in M$ **faire**
  $c_\gamma \leftarrow$ nombre d'occurrences de γ dans M ;
  si c_γ modulo $2 = 1$ **alors**
    $P \leftarrow P \cup \{\gamma\}$
  fin si
fin pour
Retourner P

$\mathbf{\Gamma}S(x)$: générateur d'ensembles

$M \longleftarrow \Gamma M(x)$;
Retourner MSet2PSet(M) ;

Algorithme générique 2: $\mathcal{S} = \mathbf{PSet}(\mathcal{A})$

Chapitre 5

Générateurs de Boltzmann pour des cycles

On va désormais s'intéresser à l'implantation de générateurs de cycles. La première classe d'objets que nous traiterons dans ce chapitre est celle des colliers. Ce premier exemple très simple nous permet d'exposer le schéma de générations des cycles. Nous verrons ensuite, avec des objets plus complexes, que le même schéma s'applique aisément pour toutes les familles de cycles. Nous étudierons donc ici, quatres générateurs :

- Un générateur de colliers binaires qui nous permet de présenter le schéma général pour les générateurs de cycles,
- Un générateur de compositions circulaires,
- Un générateur de mobiles, ainsi qu'une discussion sur le calcul de l'oracle et le choix du paramètre,
- Un générateur de graphes fonctionnels qui associe les constructions de cycle et de multi-ensemble.

5.1 Colliers bicolores

En combinatoire, ce qu'on appelle collier n-aire est en fait un mot cyclique sur un alphabet à n lettres. On peut donc décrire l'ensemble des colliers bicolores (ou binaires) par la décomposition suivante :

$$\mathcal{N} = \textsc{Cyc}(\mathcal{A} + \mathcal{B}); \quad \mathcal{A} = \mathcal{Z}; \quad \mathcal{B} = \mathcal{Z}$$

La série génératrice des colliers binaires est donc :

$$N(z) = \sum_{k=1}^{\infty} \frac{\varphi(k)}{k} \log \frac{1}{1-2z^k} \tag{5.1}$$

5.1.1 Générateur

Dans l'univers étiqueté, un cycle composé de k éléments définit k séquences différentes. Il n'en va pas de même pour des objets non étiquetés. Par exemple, le cycle de la figure 5.1 produit 5 séquences différentes et non pas 15. Cela vient du fait que certains cycles peuvent présenter des *symétries*. Pour écrire un générateur de cycles non étiquetés, il faut alors tenir compte de ces symétries. Pour ce faire, on décompose tout cycle C en une séquence de k répétitions d'un motif m irréductible, de longueur j (voir exemple, figure 5.1). On appelle k, l'*ordre de symétrie* du cycle.

La génération d'un cycle d'élément de $\{a, b\}$ se fait alors en trois étapes :

1. On détermine l'ordre de symétrie du cycle conformément à la distribution de Boltzmann. On cherche alors K, pour U aléatoire dans $[0, 1]$, tel que :

$$\frac{\sum_{k=1}^{K} \frac{\varphi(k)}{k} \log \frac{1}{1-2z^k}}{N(x)} \leq U \leq \frac{\sum_{k=1}^{K+1} \frac{\varphi(k)}{k} \log \frac{1}{1-2z^k}}{N(x)}$$

   On utilise pour cela la procédure 5.1.

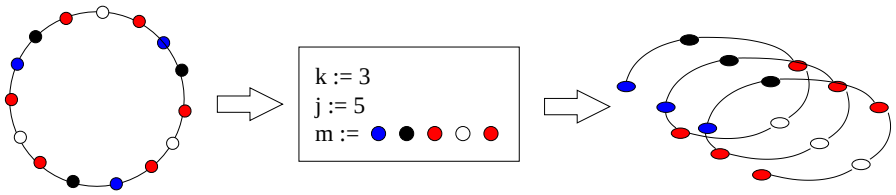


FIG. 5.1 – Décomposition d'un cycle en motifs irréductibles.

Procédure 5.1 OrdreSymetrie(x)

$U \leftarrow \text{random}(0,1)$;
$k \leftarrow 1$;
$p \leftarrow \log \frac{1}{1-2x}$;
tant que $U > \frac{p}{N(x)}$ **faire**
  $k \leftarrow k+1$;
  $p \leftarrow p + \frac{\varphi(k)}{k} \log \frac{1}{1-2x^k}$;
fin tant que
Retourner k

2. En utilisant une loi logarithmique, on tire J le nombre de composantes du motif élémentaire du cycle, tel que :

$$\mathbb{P}(J=j) = \frac{1}{\log \frac{1}{1-2x^k}} \frac{(2x^k)^j}{j}$$

3. On fabrique le motif m, i.e., une séquence de J éléments de $\{a,b\}$, engendrés grâce au générateur de Boltzmann $\Gamma L(x)$.
4. Enfin, en renvoie le cycle composé de K répétitions du motif m.

On obtient alors l'algorithme 5.1.

Algorithme 5.1 $\Gamma N(x)$: générateur de colliers binaires

procédure $\Gamma L(x)$
  si Bern$(1/2) = 1$ Retourner a **sinon** Retourner b **finsi**
fin procédure

$k \longleftarrow$ OrdreSymetrie(x) ;
$j \longleftarrow$ Loga $\left(2x^k\right)$;
$m \longleftarrow \underbrace{\Gamma L(x^k), \ldots, \Gamma L(x^k)}_{j\ fois}$;
Retourner $(\underbrace{m, \ldots, m}_{k\ fois})$;

5.1.2 Mise en œuvre

Le réglage du paramètre de ce générateur est assez simple. Comme on peut le voir sur les courbes de la figure 5.2, plus on se rapproche de la singularité $\rho = \frac{1}{2}$ de $N(x)$, plus on a de chances d'obtenir de "gros" colliers. En faisant 10000 tirages avec $x = 0.499$, on obtient des colliers pouvant aller jusqu'à ~ 3000 perles avec une distribution parfaitement conforme à la distribution théorique attendue (voir figure 5.3).

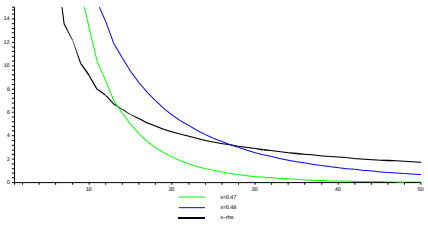


FIG. 5.2 – Distributions en taille théoriques pour les colliers bicolores

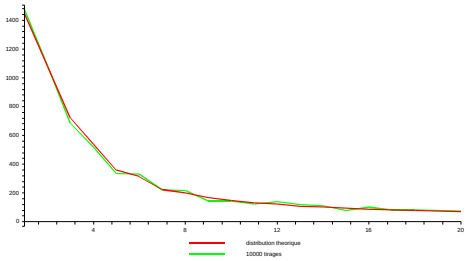


FIG. 5.3 – Distribution en taille des colliers générées par $\Gamma N(0.499)$, pour 10000 tirages.

5.2 Compositions circulaires

Classiquement, une composition est une partition où l'ordre des sommants compte. Une composition d'entier est donc une séquence d'entiers non nuls. Par extension, une composition circulaire est un cycle d'entiers non nuls, ce qui se traduit par :

$$\mathcal{C} = \textsc{Cyc}(\mathcal{Z} \times \textsc{Seq}(\mathcal{Z}))$$

La série génératrice des compositions circulaires est donc :

$$C(z) = \sum_{k=1}^{\infty} \frac{\varphi(k)}{k} \log \frac{1}{1 - \frac{z^k}{1-z^k}}$$

5.2.1 Générateur

On a vu, au chapitre 2, que la loi géométrique (Geom) fournit un générateur de Boltzmann très efficace pour les entiers non nuls. Á partir de cela, on peut construire un générateur de Boltzmann pour les compositions circulaires en suivant exactement le même schéma que pour les colliers. On peut donc directement donner l'agorithme 5.2 de génération aléatoire (où la procédure OrdreSymetrie est adaptée aux compositions circulaires).

Algorithme 5.2 $\Gamma C(x)$: générateur de compositions circulaires

$k \longleftarrow \text{OrdreSymetrie}(x)$;
$j \longleftarrow \text{Loga}\left(x^k/(1-x^k)\right)$;
$m \longleftarrow \underbrace{\text{Geom}(x^k), \ldots, \text{Geom}(x^k)}_{j\ fois}$;
Retourner $(\underbrace{m, \ldots, m}_{k\ fois})$;

Cet algorithme a une complexité $O(m)$ où m est le nombre de sommants de la composition circulaire obtenue. Or, l'ordre de grandeur du nombre moyen de sommants d'une composition circulaire de taille n est $n/2$ (voir [3], discussion sur les compositions d'entier, chapitre III.2). L'algorithme que nous proposons a donc une complexité $O(n)$.

5.2.2 Mise en œuvre

Comme pour les colliers, $C(z)$ possède une singularité évidente $\rho = \frac{1}{2}$. Plus on se rapproche de ρ et plus la probabilité d'engendrer des compositions de grande taille est forte. On peut ainsi engendrer une composition circulaire de taille 10^6 en quelques minutes alors qu'une composition circulaire engendrée par Combstruct ne peut pas dépasser une taille de 10000. La figure 5.4 nous montre par exemple une composition de taille ~ 2000 engendrée par $\Gamma C(0.49999)$. On peut voir sur la figure 5.5 la comparaison entre la distribution attendue et la distribution effective pour 5000 tirages de $\Gamma C(0.49999)$.

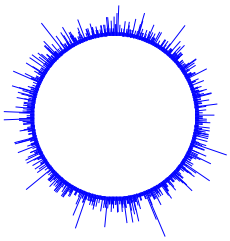


FIG. 5.4 – Une composition circulaire de taille ~ 2000.

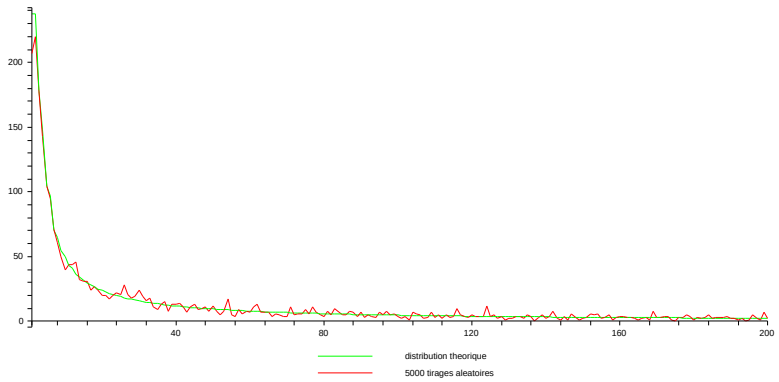


FIG. 5.5 – Distribution en taille des compositions circulaires engendrées par $\Gamma C(0.49999)$, pour 5000 tirages.

5.3 Mobiles

Un mobile est un arbre non planaire dont les noeuds ne sont plus des ensembles d'arbres, mais des cycles d'arbres. On peut donc les décrire par la relation :

$$\mathcal{W} = \mathcal{Z} + \mathcal{Z} \times \textsc{Cyc}(\mathcal{W})$$

Leur série génératrice est donc :

$$W(z) = z + z \sum_{k=1}^{\infty} \frac{\varphi(k)}{k} \log \frac{1}{1 - W(z^k)} \tag{5.2}$$

5.3.1 Générateur

Pour engendrer un mobile, on engendre soit une feuille $\square$, soit le produit d'un noeud et d'un cycle de mobiles $\bigstar\langle\ \rangle$. Pour engendrer le cycle de mobiles, la marche à suivre est encore la même que précédement. On obtient donc le générateur décrit par l'algorithme 5.3.

5.3.2 Mise en œuvre

Comme pour les autres familles d'arbres, pour obtenir des mobiles de grande taille, on choisit le paramètre du générateur proche de la singularité.

La méthode utilisée pour déterminer la singularité de $W(z)$ est celle que nous avons déjà utilisée, exposée dans [3] (chapitre VII.2.). Nous présentons ici les principales étapes de calcul. Pour des valeurs de x quelconques, on peut utiliser l'équation (5.2) tronquée pour obtenir une bonne approximation de $W(x)$. Mais pour des valeurs proches de la singularité, on commet une erreur assez importante et il faut procéder différemment. À partir de l'équation (5.2), on

Algorithme 5.3 $\Gamma W(x)$: générateur de mobiles

procédure $\Gamma CW(x)$
  $k \longleftarrow$ OrdreSymetrie(x) ;
  $j \longleftarrow$ Loga $W(x^k)$;
  $m \longleftarrow \underbrace{\Gamma W(x^k), \ldots, \Gamma W(x^k)}_{j\ fois}$;
  Retourner $\bigstar\langle \underbrace{m, \ldots, m}_{k\ fois} \rangle$;
fin procédure

si Bern $\left(\frac{x}{W(x)}\right) = 1$ **alors**
  Retourner $\square$;
sinon
  Retourner $\Gamma CW(x)$;
fin si

calcule les $W(x^k)$ récursivement depuis $k = 2$ et on utilise la fonction CayleyT[1] pour calculer le premier terme :

$$\begin{aligned}
W(x) &= x + x\log\frac{1}{1-W(x)} + \underbrace{x\sum_{k\geq 2}\frac{\varphi(k)}{k} log \frac{1}{1-W(x^k)}}_{\substack{S \\ calculé\ récursivement}} \\
\frac{W(x)}{x} &= x\log\frac{1}{1-\frac{xW(x)}{x}} + \underbrace{S+1}_{T} \\
\exp\frac{W(x)}{x} &= \frac{1}{1-\frac{xW(x)}{x}}\exp T \\
\frac{\exp(T)}{x} &= \frac{1-W(x)}{x}\exp\frac{W(x)}{x} \\
\frac{\exp\left(T-\frac{1}{x}\right)}{x} &= \frac{1-W(x)}{x}\exp\left(-\frac{1-W(x)}{x}\right) \\
\text{CayleyT}\left(\frac{\exp T-\frac{1}{x}}{x}\right) &= \frac{1-W(x)}{x} \\
W(x) &= -x\ \text{CayleyT}\left(\frac{\exp((-1+S+x)/x)}{x}\right)+1 \\
W(x) &= x\ \text{LambertW}\left(-\frac{\exp((-1+S+x)/x)}{x}\right)+1
\end{aligned}$$

La fonction CayleyT possède une singularité en e^{-1}, donc pour déterminer

[1] en Maple, on dispose de LambertW définie par $\text{CayleyT}(x) = -\text{LambertW}(-x)$

la singularité ρ de $W(z)$, on résoud

$$\frac{\exp(s+\rho-1)/\rho}{\rho} = e^{-1}$$

et on obtient $\rho \sim 0.306187516$. Avec ces données, on peut faire des tirages en la singularité et obtenir des mobiles de grande taille (jusqu'à 10^6 en quelques minutes). La figure 5.6 présente par exemple un mobile de taille ~ 500 et la figure 5.7 présente la distribution en taille obtenue pour 10000 tirages de $\Gamma W(0.306187)$ (comparée à la distribution attendue pour cette valeur de paramètre).

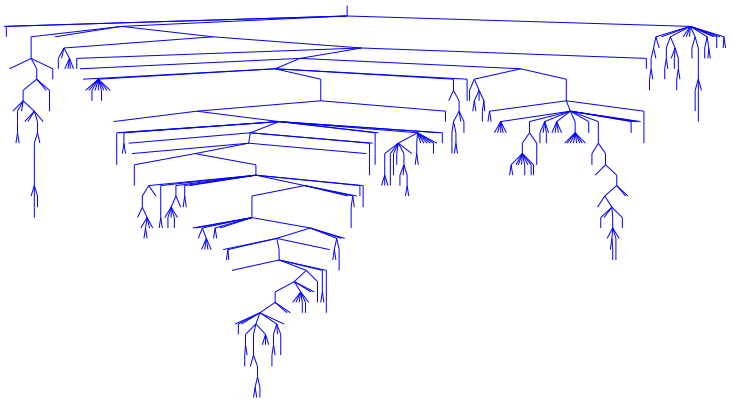


FIG. 5.6 – Un mobile de taille ~ 500 engendrée par $\Gamma W(\rho)$

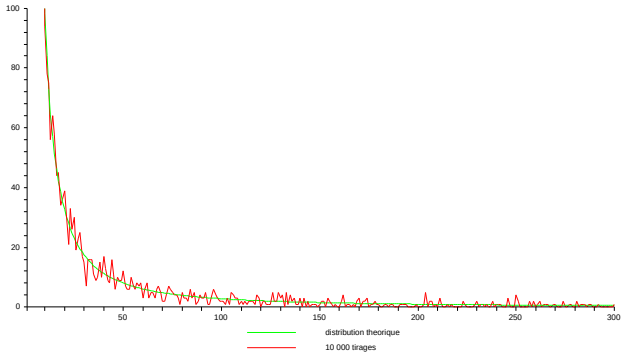


FIG. 5.7 – Distribution en taille des mobiles engendrés par $\Gamma W(0.306187)$, pour 10 000 tirages.

5.4 Graphes fonctionnels

Un graphe fonctionnel est un graphe orienté (non nécessairement connexe) dont les sommets ont un unique successeur. Celà se traduit par un multi-ensemble non étiqueté de cycles d'arbres généraux non planaires. On peut donc décrire la classe des graphes fonctionnels par les deux relations suivantes :

$$\mathcal{M} = \textsc{MSet}(\textsc{Cyc}(\mathcal{G})) \tag{5.3}$$
$$\mathcal{G} = \mathcal{Z} \times \textsc{MSet}(\mathcal{G}) \tag{5.4}$$

La série génératrice des graphes fonctionnels est donc la suivante :

$$M(z) = \exp\left(\sum_{k\geq 1} \frac{CT(z^k)}{k}\right) \tag{5.5}$$
$$CT(z) = \sum_{k\geq 1} \frac{\varphi(k)}{k} \log \frac{1}{1-T(z^k)} \tag{5.6}$$
$$T(z) = z \exp\left(\sum_{k\geq 1} \frac{T(z^k)}{k}\right) \tag{5.7}$$

5.4.1 Générateur

On réutilise le générateur d'arbre généraux non planaires $\Gamma T(x)$ proposé à la section 3.2.
$\Gamma M(x)$ se décompose alors ainsi :

- On implante un générateur de cycles d'arbres $\Gamma CT(x)$,
- On engendre un multi-ensemble d'objets générés par $\Gamma CT(x)$.

Cela donne le générateur décrit par l'agorithme 5.4, où $CT(z)$ est la série génératrice des cycles d'arbres.

5.4.2 Mise en œuvre

La singularité de $M(z)$ est la même que celle des arbres généraux. Et c'est en la singularité qu'on peut engendrer des arbres de grande taille. On réutilise donc les valeurs obtenues à la section 3.2.2 pour calculer les oracles utilisés par ΓM.

On peut, par exemple, voir à la figure 5.8, un graphe fonctionnel de taille ~ 200 et à la figure 5.9 la distribution en taille obtenue pour 10000 tirages de $\Gamma M(0.33)$.

Conclusion

Nous disposons désormais d'un algorithme efficace (voir Algorithme générique 3) de génération de cycles sous modèle de Boltzmann. Les générateurs qu'il fournit

Algorithme 5.4 $\Gamma M(x)$: générateur de graphes fonctionnels

procédure $\Gamma CT(x)$
  $k \longleftarrow$ OrdreSymetrie(x) ;
  $j \longleftarrow$ Loga $\big(T(x^k)\big)$;
  $m \longleftarrow$ ($\underbrace{\Gamma T(x^k), \ldots, \Gamma T(x^k)}_{j\ fois}$) ;
  Retourner $\bigstar\langle\ \underbrace{m, \ldots, m}_{k\ fois}\ \rangle$;
fin procédure

procédure $\Gamma CT_k(x)$
  $c \longleftarrow \Gamma CT(x)$;
  Retourner $\underbrace{c, \ldots, c}_{k\ fois}$;
fin procédure

procédure $\mathbf{\Gamma} S_k(x)$
  $p \leftarrow$ Pois $\left(\frac{CT(x^k)}{k}\right)$;
  Retourner $\underbrace{\Gamma CT_k(x^k)\ ;\ \ldots\ ;\ \Gamma CT_k(x^k)}_{p \text{ fois}}$
fin procédure

procédure $\mathbf{\Gamma} S_{\geq 1,k}(x)$
  $p \leftarrow$ Pois$_{\geq 1}$ $\left(\frac{CT(x^k)}{k}\right)$;
  Retourner $\underbrace{\Gamma CT_k(x^k)\ ;\ \ldots\ ;\ \Gamma CT_k(x^k)}_{p \text{ fois}}$
fin procédure

$k \leftarrow$ IndiceMaximalPaquet(x)
si $k = 0$ **alors**
  Retourner $\emptyset$;
sinon
  Retourner ($\Gamma S_1(x)\ ;\ \ldots\ ;\ \Gamma S_{k-1}(x)\ ;\ \Gamma S_{\geq 1,k}(x)$)
fin si

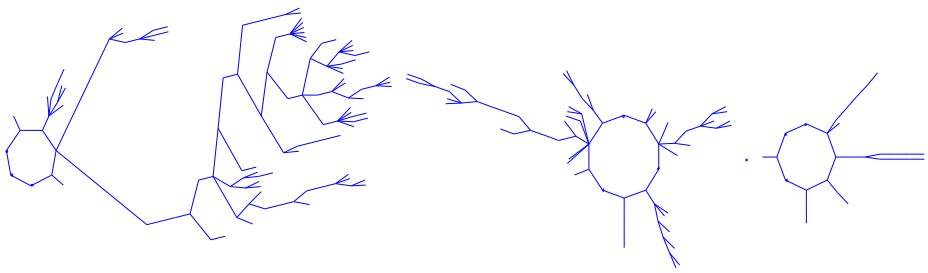


FIG. 5.8 – Un graphe fonctionnel de taille ~ 200 engendrée par $\Gamma M(\rho - 0.0001)$

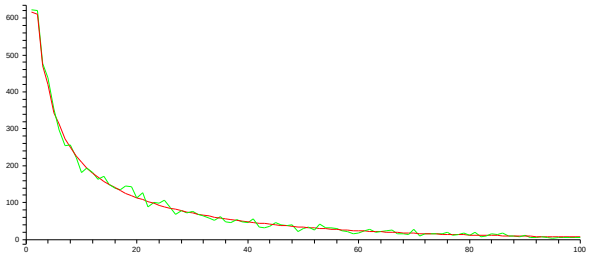


FIG. 5.9 – Distribution en taille des graphes fonctionnels engendrés par $\Gamma M(0.33)$, pour 10000 tirages.

ont une complexité $O(n)$ dans le pire des cas. Ces résultats se traduisent par les théorèmes 5.1 et 5.2. .

Théorème 5.1 *Si on possède un générateur de Boltzmann pour $\mathcal{A}$, alors, l'algorithme générique 3 fournit un générateur aléatoire de Boltzmann pour $\mathcal{C} =$ CYC($\mathcal{A}$). Cet algorithme produit un objet $\mu \in \mathcal{M}$ en temps linéaire en la taille de μ.*

Théorème 5.2 *Pour les classes combinatoires suivantes, pour des valeurs du paramètre x optimisées, et en utilisant une méthode par rejet, on peut obtenir des générateurs de Boltzmann en taille approchée avec les complexités moyennes suivantes :*

Colliers binaires de taille n	$O(n)$
Compositions circulaires de taille n	$O(n)$
Mobiles de taille n	$O(n)$
Graphes fonctionnels de taille n	$O(n)$

– Série génératrice

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

– Générateur de Boltzmann pour $\mathcal{C}$

1. tirer k, l'ordre de symetrie du cycle avec la procédure suivante :

   OrdreSymetrie(x)

   $U \leftarrow \text{random}(0, 1)$;
   $k \leftarrow 1$;
   $p \leftarrow \log \frac{1}{1-A(x)}$;
   tant que $U > \frac{p}{C(x)}$ **faire**
      $k \leftarrow k + 1$;
      $p \leftarrow p + \frac{\varphi(k)}{k} \log \frac{1}{1-A(x^k)}$;
   fin tant que
   Retourner k

2. tirer $j \geq 1$, le nombre de composantes du motif élémentaire de la séquence, tel que

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

3. tirer m, le motif élémentaire de la séquence, composé de j éléments de $\mathcal{A}$
4. renvoyer le cycle représenté par la séquence constituée de k fois le motif m.

$\Gamma C(x)$: générateur de cycles

$k \longleftarrow \text{OrdreSymetrie}(x)$;
$j \longleftarrow \text{Loga}\big(A(x^k)\big)$;
$m \longleftarrow \underbrace{\Gamma A(x^k), \ldots, \Gamma A(x^k)}_{j\ fois}$;
Retourner $(\underbrace{m, \ldots, m}_{k\ fois})$;

Algorithme générique 3: $\mathcal{C} = \mathbf{Cyc}(\mathcal{A})$

Chapitre 6

Conclusion

Nous venons de développer une théorie générale qui permet de construire, de façon systématique, des générateurs aléatoires efficaces pour un large ensemble de classes combinatoires. Pour les trois constructions MSET, PSET et CYC, nous avons donné des algorithmes génériques détaillés de générateurs de Boltzmann (pages 40, 55 et 68). Nous pouvons donc énoncer le théorème suivant :

Théorème 6.1 *Il existe un procédé de construction effectif qui, partant d'une classe combinatoire $\mathcal{C}$ quelconque d'objets étiquetés, spécifiée au moyen des constructions suivantes :*

$$\{\varepsilon, \mathcal{Z}, +, \times, \text{SEQ}, \text{MSET}, \text{PSET}, \text{CYC}\}$$

produit un générateur aléatoire conforme au modèle de Boltzmann.

Pour la construction d'ensemble, dans le cas où le rayon de convergence de la série $C(z)$ est < 1, le générateur ainsi élaboré a un coût linéaire en la taille de l'objet engendré, en moyenne.

Pour les autres constructions, le générateur à un coût linéaire en la taille de l'objet engendré, dans le pire des cas.

De plus, nous avons implanté un certain nombre d'algorithmes spécifiques à la génération d'objets appartenant à des classes combinatoires données. Nous avons effectué des réglages précis sur les paramètres de ces générateurs, dans le but d'obtenir des objets de grande taille. Nous avons pu, dès lors, engendrer des objets de tailles plusieurs dizaines de fois supérieures aux limites qu'atteint la bibliothèque Combstruct de MAPLE, qui implante des algorithmes basés sur la méthode récursive, présentée au chapitre 1. Nous avons notamment été en mesure d'engendrer des partitions de taille $\sim 10^{10}$. Ces résultats se traduisent par le théorème suivant :

Théorème 6.2 *Pour les familles de classes combinatoires suivantes, pour des valeurs du paramètre x optimisées, et en utilisant une méthode par rejet, on peut obtenir des générateurs de Boltzmann en taille approchée, ayant les complexités moyennes suivantes :*

Partitions d'entier	$O(\sqrt{n})$
Arbres non planaires	$O(n)$
Langages finis	$O(n)$
Colliers	$O(n)$
Compositions circulaires	$O(n)$
Mobiles	$O(n)$
Graphes fonctionnels	$O(n)$

Nous envisageons désormais de poursuivre ce travail par l'implantation d'une bibliothèque générique du type Combstruct pour l'ensemble des constructions de la spécification donnée par le théorème 6.1.

Nous projetons d'enrichir encore l'ensemble des classes combinatoires couvertes par cette théorie, en approfondissant les résultats que nous obtenons pour les constructions avec contraintes de cardinalité.

Nous souhaitons également élargir les champs d'applications des générateurs de Boltzmann, dans des domaines tels que le test de logiciels et la bio-informatique, en offrant, par exemple, la possibilité d'imposer des contraintes sur les objets engendrés.

Chapitre 7

Annexes

7.1 Validité du générateur de multi-ensembles[1]

Pour montrer que l'algorithme $\Gamma M(x)$ est bien un générateur de Boltzmann pour la classe $MSet(\mathcal{C})$, nous aurons besoin de deux lemmes techniques et d'un lemme intermédiaire :

Lemme 7.1 *Soit λ un réel positif et $(X_k)_{k\geq 1}$ une suite de variables aléatoires indépendantes tels que pour tout $k \geq 1$, X_k suit une loi de Poisson de paramètre $\frac{\lambda^k}{k}$. Alors la variable aléatoire $X := \sum_k kX_k$ suit une loi géométrique de paramètre λ.*

Preuve. Comme X_k suit une loi de Poisson de paramètre $\frac{\lambda^k}{k}$, la fonction caractéristique de X_k, définie par $F_k(u) := \sum_i P(X_k = i)u^i$, vaut $\exp(\frac{\lambda^k}{k}(u-1))$. En conséquence, la fonction caractéristique de kX_k vaut $\exp(\frac{\lambda^k}{k}(u^k-1))$. Comme les variables aléatoires X_k sont indépendantes, la fonction caractéristique $F_X(u)$ de X est telle que $F_X(u) = \prod_{k=1}^{\infty} \exp(\frac{\lambda^k}{k}(u^k-1))$. Donc on a $F_X(u) = \exp(\sum_{i=1}^{\infty} \frac{(\lambda u)^k}{k} - \frac{\lambda^k}{k}) = \exp(\log(\frac{1-\lambda}{1-\lambda u})) = \frac{1-\lambda}{1-\lambda u} = \sum_k (1-\lambda)\lambda^k u^k$. En conséquence, $P(X = k) = (1-\lambda)\lambda^k$, i.e., X suit une loi géométrique de paramètre λ.

Definition : Soit $(p_1, \ldots, p_l)$ un vecteur de probabilité, i.e., $p_1 + \ldots + p_l = 1$. Un vecteur aléatoire Y de dimension l est dit avoir une loi de Bernoulli de paramètre $(p_1, \ldots, p_l)$ si on a : $P(Y = (1, 0, \ldots, 0)) = p_1$, $P(Y = (0, 1, \ldots, 0)) = p_2$,..., $P(Y = (0, \ldots, 0, 1)) = p_l$.

Le lemme suivant, bien connu, exprime qu'une loi de Poisson composée avec une loi de Bernoulli se distribue en des lois de Poisson indépendantes.

Lemme 7.2 *Soit $(p_1, \ldots, p_l)$ un vecteur de probabilité et soit λ un réel positif. Soit $(Y_1)_{i\geq 1}$ une séquence de vecteurs aléatoires indépendants suivant des lois de Bernoulli de paramètre $(p_1, \ldots, p_l)$. Soit N une loi de Poisson de paramètre λ indépendante des Y_i. Alors le vecteur aléatoire $Z = (Z_1, \ldots, Z_l)$ défini par $Z := \sum_{i=1}^{N} Y_i$ est tel que $Z_1 \in Pois(\lambda p_1), Z_2 \in Pois(\lambda p_2), \ldots, Z_l \in Pois(\lambda p_l)$. De plus les variables $Z_1, Z_2, \ldots, Z_l$ sont indépendantes.*

Preuve. Par définition d'une loi de Bernoulli $Y = (Y_1, \ldots, Y_l)$ de paramètre $(p_1, p_2, \ldots, p_l)$, sa fonction caractéristique, définie par $\sum_{i_1,\ldots,i_l} P(Y_1 = i_1, \ldots, Y_l = i_l)u_1^{i_1}u_2^{i_2}\cdots u_l^{i_l}$, vaut $p_1u_1 + \ldots + p_lu_l$. Donc, pour N fixé et Y_i des lois de Bernoulli de paramètre $(p_1, \ldots, p_l)$ indépendantes, la fonction caractéristique de $\sum_{i=1}^{N} Y_i$ vaut $(p_1u_1 + \ldots + p_lu_l)^N$. Si N suit une loi de Poisson de paramètre λ indépendant des Y_i, alors la fonction génératrice $\phi_Z(u_1, \ldots, u_l)$ de $Z := \sum_{i=1}^{N} Y_i$ vérifie

[1] proposé par Eric Fusy

$$\begin{aligned}\phi_Z(u_1,\ldots,u_l) &= \sum_{j=0}^{\infty} e^{-\lambda}\frac{\lambda^j}{j!}(p_1u_1+\ldots+p_lu_l)^j \\ &= e^{-\lambda}\exp(\lambda(p_1u_1+\ldots+\lambda p_lu_l)) \\ &= \prod_{i=1}^{l}\exp(\lambda p_i(u_i-1))\end{aligned}$$

Comme une loi de Poisson de paramètre λ a pour fonction caractéristique $u \to e^{\lambda(u-1)}$, on déduit directement de l'expression de $\phi_Z(u_1,\ldots,u_l)$ que les composantes $Z_1,\ldots,Z_l$ de Z sont indépendantes et que $Z_1 \in Pois(\lambda p_1),\ldots,Z_l \in Pois(\lambda p_l)$.

Lemme 7.3 *Soit $A = (\gamma^{(1)},\ldots,\gamma^{(m)})$ un ensemble fini d'objets de $\mathcal{C}$. Pour $1 \leq i \leq m$, soit $X^{(i)}$ la variable aléatoire donnant le nombre de copies de $\gamma^{(i)}$ dans un mutli-ensemble tiré avec $\Gamma M(x)$. Soit $\overline{X}$ la variable aléatoire donnant le nombre de composantes qui ne sont pas dans A pour un multi-ensemble tiré avec $\Gamma M(x)$ Alors on a les propriétés suivantes :*

- *Les variables $(X^{(1)},\ldots,X^{(m)},\overline{X})$ sont indépendantes.*
- *Pour $1 \leq i \leq m$, $X^{(i)}$ suit une loi géométrique de paramètre $x^{|\gamma^{(i)}|}$.*
- *La variable aléatoire $\overline{X}$ vérifie $P(\overline{X}=0) = \prod_{\gamma\in A}(1-x^{|\gamma|})$.*

Preuve. Pour $k \geq 1$ et $1 \leq i \leq m$, on note $X_k^{(i)}$ la variable aléatoire donnant le nombre d'instances de k-paquets de l'objet $\gamma^{(i)}$ tirés par $\Gamma M_k(x)$. On note aussi $\overline{X}_k$ la v.a. donnant le nombres de k-paquets d'objets qui ne sont pas dans A lors d'un tirage de ΓM_k. Par construction de $\Gamma M_k(x)$, le vecteur $(X_k^{(1)},\ldots,X_k^{(m)},\overline{X}_k)$ vérifie les conditions du lemme 7.2, avec $\lambda = \frac{C(x^k)}{k}$ et $(p_1,\ldots,p_{m+1}) = \left(\frac{x^{k|\gamma^{(1)}|}}{C(x^k)},\ldots,\frac{x^{k|\gamma^{(m)}|}}{C(x^k)},\frac{\sum_{\gamma\in A}x^{k|\gamma|}}{C(x^k)}\right)$. En conséquence, $X_k^{(1)} \in Pois\left(\frac{x^{k|\gamma^{(1)}|}}{k}\right),\ldots,X_k^{(m)} \in Pois\left(\frac{x^{k|\gamma^{(m)}|}}{k}\right),\overline{X}_k \in Pois\left(\frac{\sum_{\gamma\in A}x^{k|\gamma|}}{k}\right)$ et les v.a. $X_k^{(1)},\ldots,X_k^{(m)},\overline{X}_k$ sont indépendantes. Par construction de $\Gamma M(x)$ à partir de la séquence de procédures $\Gamma M_k(x)$, pour $1 \leq i \leq m$ la v.a. $X^{(i)}$ donnant le nombre de copies de $\gamma^{(i)}$ pour un tirage de $\Gamma M(x)$ vérifie $X^{(i)} = \sum_k kX_k^{(i)}$. De même la v.a. $\overline{X}$ donnant le nombre de composantes qui ne sont pas dans A vérifie $\overline{X} = \sum_k k\overline{X}_k$. En utilisant le fait que les v.a. $X_k^{(1)},\ldots,X_k^{(m)},\overline{X}_k$ sont indépendantes et que les processus $(\Gamma M_k)_{k\geq 1}$ sont indépendants entre eux, on conclut que les v.a. $X^{(1)},\ldots,X^{(m)},\overline{X}$ sont indépendantes. Ensuite, en utilisant le lemme 7.1, on conclut que pour $1 \leq i \leq m$, $X^{(i)}$ suit une loi géométrique de

paramètre $x^{|\gamma^{(i)}|}$. En outre, on a :

$$\begin{aligned} Pr(\overline{X}=0) &= \prod_{k=1}^{\infty} Pr(\overline{X}_k=0) = \prod_{k=1}^{\infty} \exp\left(-\frac{\sum_{\gamma\in A} x^{k|\gamma|}}{k}\right) \\ &= \exp\left(-\sum_{k=1}^{\infty}\frac{1}{k}\sum_{\gamma\in A} x^{k|\gamma|}\right) = \exp\left(-\sum_{\gamma\in A}\sum_{k=1}^{\infty}\frac{x^{k|\gamma|}}{k}\right) \\ &= \prod_{\gamma\in A}\exp\left(-\log\left(\frac{1}{1-x^{|\gamma|}}\right)\right) = \prod_{\gamma\in A}(1-x^{|\gamma|}) \end{aligned}$$

Proposition 7.1 *Le générateur $\Gamma M(x)$ est un générateur de Boltzmann pour la classe $MSet(\mathcal{C})$.*

Preuve. Soit $M \in MSet(\mathcal{C})$. Soit n la taille de M et soit $Pr(M)$ la probabilité que M soit tiré par $\Gamma M(x)$. Nous allons montrer que $Pr(M) = \frac{x^n}{M(x)}$. On note $A := (\gamma^{(1)}, \ldots, \gamma^{(m)})$ l'ensemble des objects de $\mathcal{C}$ dont le nombre de composantes dans M est non nul. Pour $1 \leq i \leq m$, on note $c_i > 0$ ce nombre de composantes. Remarquons que $n = \sum_{i=1}^{m} c_i|\gamma^{(i)}|$. En utilisant le lemme 7.3, on a directement : $Pr(M) = \prod_{i=1}^{m}(1-x^{|\gamma^{(i)}|})x^{|\gamma^{(i)}|c_i}\prod_{\gamma\in A}(1-x^{|\gamma|}) = \frac{x^n}{\prod_{\gamma\in\mathcal{C}}\frac{1}{1-x^{|\gamma|}}} = \frac{x^n}{M(x)}$.

7.2 Validité du générateur de cycles

7.2.1 Première approche[2]

Soit la classe $\mathcal{C} = \textsc{Cyc}(\mathcal{A})$ des cycles d'objets de $\mathcal{A}$ et sa série génératrice associée, $C(z)$:

$$C(z) = \sum_{k\geq 1}\frac{\varphi(k)}{k}\log\frac{1}{1-A(z^k)} \tag{7.1}$$

$$= \sum_{k,l\geq 1}\frac{\varphi(k)}{k}\frac{A(z^k)^l}{l} \tag{7.2}$$

Le générateur, de paramètre x, que nous proposons :

– choisit K aléatoire tel que :

$$Pr(K=k) = \frac{1}{C(x)}\frac{\varphi(k)}{k}\log\frac{1}{1-A(x^k)}$$

– puis tire L tel que :

$$Pr(L=l) = \frac{A(x^k)^l}{l}\frac{1}{\log(1-A(x^k))^{-1}}$$

[2]avec l'aide de Philippe Flajolet

– produit un cycle [$\underbrace{u, u, \ldots, u}_{k\ fois}$] de longueur $m = kl$ avec k répétitions de u, un l-uplet de $\Gamma A(x^k)$.

On veut montrer que ce générateur suit le modèle de Boltzmann, i.e., qu'il engendre tout objet γ tel que $|\gamma| = n$ avec une probabilité :

$$\frac{x^n}{C(x)}$$

On observe que :

$$Pr(K = k, L = l) = \frac{\varphi(k)}{kl} \frac{A(x^k)^l}{C(x)}$$

Pour cette preuve, on utilisera la propriété suivante :

$$\sum_{k|m} \varphi(k) = m$$

On distingue trois cas :

1. Soit un cycle $\gamma = (\ \underbrace{\alpha, \ldots, \alpha}_{m\ fois}\ )$, avec $|\alpha| = a$ et $|\gamma| = n$.
   Il est engendré avec une probabilité :

$$P(\gamma) = \sum_{kl=m} \frac{\varphi(k)}{kl} \frac{A(x^k)^l}{C(x)} \left(\frac{x^{ka}}{A(x^k)} \right)^l \tag{7.3}$$

$$= \frac{1}{m} \frac{x^n}{C(x)} \sum_{k|m} \varphi(k) \tag{7.4}$$

$$= \frac{x^n}{C(x)} \tag{7.5}$$

2. Soit un cycle $\gamma = (\ \alpha_1, \ldots, \alpha_m\ )$ qui est primitif (i.e. différent de tous ses conjugués), avec $|\gamma| = n$.
   Il est engendré avec une probabilité :

$$P(\gamma) = m \sum_{kl=m} \frac{\varphi(1)}{m} \frac{x^{|\alpha_1|+\cdots+|\alpha_m|}}{C(x)} \tag{7.6}$$

$$= \frac{x^n}{C(x)} \tag{7.7}$$

3. Soit un cycle $\gamma = (\ \underbrace{u, \ldots, u}_{r\ fois}\ )$, avec $u = [u_1, \ldots, u_s]$ primitif.
   Il peut être engendré comme une séquence de k répétitions d'un cycle

primitif, avec $k|r$.
Il est donc engendré avec une probabilité :

$$P(\gamma) = s \sum_{\substack{kl=m \\ k|r}} \frac{\varphi(k)}{kl} \frac{x^n}{C(x)} \tag{7.8}$$

$$= \frac{s}{m} \frac{x^n}{C(x)} \sum_{k|r} \varphi(k) \tag{7.9}$$

$$= \frac{sr}{m} \tag{7.10}$$

$$= \frac{x^n}{C(x)} \tag{7.11}$$

7.2.2 Deuxième approche[3]

Cycles de longueur k

Soit $\mathcal{A}$ une classe combinatoire d'objets, de série génératrice $A(x)$. On note $\mathcal{C}_k$ l'ensemble des cycles d'objets de $\mathcal{A}$ de longueur k. Défini rigoureusement, $\mathcal{C}_k$ est l'ensemble des séquences de longueur k d'objets de $\mathcal{A}$ modulo décalage cyclique. Autrement dit, $\mathcal{C}_k$ est l'ensemble des orbites de séquences de longueur k d'objets de $\mathcal{A}$ sous l'action, dite de décalage, du groupe $\mathbf{Z}/k\mathbf{Z}$, l'action de $l \in \mathbf{Z}/k\mathbf{Z}$ sur $(\gamma_1, \ldots, \gamma_k)$ donnant $(\gamma_{1+l}, \ldots, \gamma_k, \gamma_1, \ldots, \gamma_l)$.

Le lemme de Burnside permet de traiter combinatoirement un ensemble défini modulo l'action d'un groupe. Plus précisément, il donne une formule pour compter les orbites d'un ensemble fini E sur lequel agit un groupe fini G. Pour $g \in G$, on note Fix_g l'ensemble des éléments de E que laisse invariants l'action de g. On note Orb_E l'ensemble des orbites de E sous l'action de G. La formule de Burnside est la suivante :

$$|Orb_E| = \frac{1}{|G|} \sum_{g \in G} |Fix_g|.$$

Cette formule découle du fait que la projection naturelle de $\cup_{g \in G} Fix_g$ sur Orb_E est telle que chaque élément de Orb_E a exactement k antécédants.

Nous introduisons maintenant une définition qui nous permet de formuler le lemme de Burnside de manière agréable pour le cas des cycles de longueur k : on appelle *séquence de décalage* un couple formé d'une séquence $(\gamma_1, \ldots, \gamma_k)$ d'objets de $\mathcal{A}$ et d'un élément $l \in \mathbf{Z}/k\mathbf{Z}$ tel que $(\gamma_{1+l}, \ldots, \gamma_k, \gamma_1, \ldots, \gamma_l) = (\gamma_1, \ldots, \gamma_k)$. L'élément l est appelé *élément fixateur* de la séquence de décalage.

Proposition 7.2 *Pour tout $\nu \in \mathcal{C}_k$, il existe exactement k séquences de décalage dont le cycle associé est ν.*

[3] proposée par Eric Fusy

La proposition 7.2 a la conséquence suivante : pour réaliser un générateur de Boltzmann pour $\mathcal{C}_k$, il suffit de réaliser un générateur de Boltzmann pour les séquences de décalage de longueur k.

Étudions maintenant la structure combinatoire des séquences de décalage. Pour $l \in \mathbf{Z}/k\mathbf{Z}$, on note $\mathcal{A}_{k,l}$ l'ensemble des séquences de décalage de longueur k ayant l pour élément fixateur. Comme l'action de 0 laisse fixe toute séquence, l'ensemble $\mathcal{A}_{k,0}$ s'identifie à l'ensemble $\mathcal{A}_k$ des séquences de longueur k d'objets de $\mathcal{A}$. En revanche, si $l \neq 0$, le fait que l fixe une séqence de $\mathcal{A}_k$ indique une symétrie de la séquence. Précisément, soit $l \in \mathbf{Z}/k\mathbf{Z}$ et soit r l'ordre de l dans $\mathbf{Z}/k\mathbf{Z}$. Alors une suite $(\gamma_1, \ldots, \gamma_k) \in \mathcal{A}_{k,l}$ est caractérisée par le fait qu'elle s'écrit comme la concaténation de r copies d'une suite $(\alpha_1, \ldots, \alpha_{k/r}) \in \mathcal{A}_{k/r}$. En notant $\mathcal{A}_j^{(r)}$ l'ensemble des séquences de $\mathcal{A}_{jr}$ qui peuvent s'écrire comme la concaténation de r copies d'une séquence de $\mathcal{A}_j$, on a donc

$$\mathcal{A}_{k,l} \equiv \mathcal{A}_{k/r}^{(r)} \tag{7.12}$$

où r est l'ordre de l dans $\mathbf{Z}/k\mathbf{Z}$.

La série génératrice de $\mathcal{A}_{k,l}$, notée $A_{k,l}(x)$, est donc égale à $A(x^r)^{k/r}$ et on a le générateur de Boltzmann suivant pour $\mathcal{A}_{k,l}$:

$\Gamma A_j^{(r)}(x)$: 1) $\gamma \leftarrow (\Gamma A(x^r), \ldots, \Gamma A(x^r))$ $\{j$ appels indépendants$\}$
2) retourner $(\gamma, \ldots, \gamma)$ $\{r$ copies$\}$

$\Gamma A_{k,l}(x)$: 1) $r \leftarrow$ ordre de l dans $\mathbf{Z}/k\mathbf{Z}$
2) retourner $\Gamma A_{k/r}^{(r)}(x)$

Par un traitement arithmétique élémentaire, il est bien connu que pour chaque diviseur r de k, il existe $\phi(r)$ éléments d'ordre r dans $\mathbf{Z}/k\mathbf{Z}$, où $\phi(r) = \#\{i < r \text{ avec } pgcd(i, r) = 1\}$ est la fonction d'Euler. Par conséquent, en notant $C_k(x)$ la série génératrice de $\mathcal{C}_k$, on déduit de la proposition 7.2 et de l'équation 7.12 l'expression suivante pour $C_k(x)$:

$$C_k(x) = \frac{1}{k} \sum_{r|k} \phi(r) A(x^r)^{k/r}.$$

De plus, on obtient de manière immédiate le générateur de Boltzmann suivant pour $\mathcal{C}_k$:

$\Gamma C_k(x)$: 1) tirer un entier r parmi les diviseurs de k
sous la distribution $Pr(r) = \phi(r) A(x^r)^{k/r} / (k C_k(x))$
2) retourner $\Gamma A_{k/r}^{(r)}(x)$

Cycles généraux

Soit $\mathcal{C}$ la classe des cycles d'objets d'un ensemble $\mathcal{A}$, i.e. $\mathcal{C} = \cup_{k \geq 1} \mathcal{C}_k$. On note $C(x)$ la série génératrice de $\mathcal{C}$. Tout d'abord, nous dérivons une expression explicite pour $C(x)$ en fonction de $A(x)$:

$$\begin{aligned} C(x) &= \sum_{k\geq 1} C_k(x) \\ &= \sum_{k\geq 1} \frac{1}{k} \sum_{r,j;rj=k} \phi(r) A(x^r)^j \\ &= \sum_{r,j\geq 1} \frac{1}{rj} \phi(r) A(x^r)^j \\ &= \sum_{r\geq 1} \frac{\phi(r)}{r} \sum_{j\geq 1} \frac{A(x^r)^j}{j} \\ &= \sum_{r\geq 1} \frac{\phi(r)}{r} \log\left(\frac{1}{1-A(x^r)}\right) \end{aligned}$$

Nous décrivons maintenant un algorithme de tirage aléatoire d'un cycle sur $\mathcal{A}$, pour un réel $x \geq 0$ inférieur au rayon de convergence de $C(x)$:

$\Gamma C(x)$: 1) tirer un entier $r \geq 1$ sous la distribution
$Pr(r) = (\phi(r)/r) \cdot \log(1/(1-A(x^r)))/C(x)$
2) tirer un entier $j \geq 1$ sous la distribution
$Pr(j) = (A(x^r)^j/j)/\log(1/(1-A(x^r)))$
3) retourner $\Gamma A_j^{(r)}(x)$

Proposition 7.3 *L'algorithme $\Gamma C(x)$ est un générateur conforme au modèle de Boltzmann pour les cycles sur $\mathcal{A}$.*

Preuve : L'identité $\mathcal{C} = \cup_{k\geq 1} \mathcal{C}_k$ assure qu'un générateur de Boltzmann pour $\mathcal{C}$ s'obtient en tirant un entier $k \geq 1$ sous la distribution $Pr(k) = C_k(x)/C(x)$, puis en appelant $\Gamma C_k(x)$. Selon la proposition 7.2, cela revient à tirer un couple (k,l) sous la distribution $Pr(k,l) = A_{k,l}(x)/(kC_k(x))$, puis à appeler $\Gamma A_{k,l}(x)$. L'identité 7.12 assure que cette procédure se ramène au tirage d'un couple (r,j) sous la distribution $Pr(r,j) = (\phi(r)A(x^r)^j)/(r \cdot j \cdot C(x))$, puis à l'appel de $\Gamma A_j^{(r)}(x)$, ce que fait exactement l'algorithme $\Gamma C(x)$. □

Remarquons que l'algorithme $\Gamma C(x)$ ne correspond pas à une transposition directe de l'identité $\mathcal{C} = \cup_{k\geq 1} \mathcal{C}_k$, qui aurait été de tirer $k \geq 1$ sous la distribution $C_k(x)/C(x)$, puis de renvoyer $\Gamma C_k(x)$. L'astuce qui rend l'algorithme simple et efficace consiste à tirer l'ordre de la séquence renvoyée *avant* de tirer sa longueur.

Liste des Algorithmes

Bibliographie

[1] A. Denise, M.-C. M.-C. Gaudel, and Gouraud S.-D. A generic method for statistical testing. In *Fifteenth IEEE International Symposium on Software Reliability Engineering (ISSRE)*, pages 25–34, Saint Malo, novembre 2004.

[2] Ph. Duchon, Ph. Flajolet, G. Louchard, and G. Schaeffer. Boltzmann samplers for the random generation of combinatorial structures. *Combin. Probab. Comput.*, 13(4-5) :577–625, 2004.

[3] Ph. Flajolet and R. Sedgewick. Analytic combinatorics. (Chapters I,II,III,IV,V,VI,VII,VIII,IX*, Version of April 29, 2005), à paraître.

[4] Ph. Flajolet, P. Zimmerman, and B. Van Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theoret. Comput. Sci.*, 132(1-2) :1–35, 1994.

[5] S.-D. Gouraud. Application de la génération aléatoire de structures combinatoires au test de logiciel. In *Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL)*, pages 99–112, Nancy, juin 2001.

[6] A. Nijenhuis and H. S. Wilf. *Combinatorial algorithms*. Academic Press Inc., New York, second edition, 1978.

[7] Y. Ponty. *Etudes combinatoire et génération aléatoire des structures secondaires d'ARN*. Mémoire de dea, Université Paris Sud, 2003.

[8] J.-L. Remy. Un procédé itératif de dénombrement d'arbres binaires et son application à leur génération aléatoire. *Informatique Théorique et Applications*, 19(2) :179–195, 1985.

[9] A. M. Vershik. Statistical mechanics of combinatorial partitions, and their limit configurations. *Funct. Anal. Appl.*, 30(2) :90–105, 1996.

[10] H. S. Wilf. Three problems in combinatorial asymptotics. *J. Combin. Theory Ser. A*, 35(2) :199–207, 1983.